Karlsruhe Institute of Technology

Automated Generation of Fluent Lecture Recordings

Bachelor's Thesis of

Thomas Kiefer

At the KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Artificial Intelligence for Language Technologies (AI4LT)

First examiner:	Prof. Dr. Jan Niehues
Second examiner:	Prof. Dr. Alexander Waibel
Advisor:	M.Sc. Zhaolin Li

13. May 2023 – 13. September 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 13. September 2023

. . . .

(Thomas Kiefer)

Abstract

In the rapidly evolving educational landscape, this thesis addresses the pressing need for the generation of high-quality digital educational resources with reduced manual intervention. The research explores the capabilities of Large Language Models (LLMs) in enhancing the fluency and sentence structure of academic lecture recordings. It revolves around two central research questions: understanding the influence of different prompting techniques on LLMs in enhancing lecture recordings, and devising a method to train a style classifier that prioritizes style over content in feature extraction for dense embeddings. Leveraging models such as Vicuna and LLaMA 2, the research investigates the optimal conditions for each model, highlighting the necessity for tailored approaches in leveraging different LLMs. A detailed analysis revealed that while LLaMA 2 excelled in style transfer strength, Vicuna demonstrated superior performance in content preservation. The few-shot method, which utilized handcrafted rephrased sentences as examples, emerged as the most effective strategy for content preservation. The thesis not only offers a deep understanding of the current technologies but also presents a structured approach to the generation system. By bridging the existing gap in the integration of advancements in Natural Language Processing (NLP) into a fluent lecture generation system, this research marks a significant step towards the realization of automated systems capable of transforming lecture recordings to a level that is comparable with seasoned public speakers.

Zusammenfassung

In der sich rasch wandelnden Bildungslandschaft adressiert diese Thesis die dringende Notwendigkeit, hochwertige digitale Bildungsressourcen mit minimalem manuellen Aufwand zu erstellen. Die Arbeit erkundet die Potenziale von "Large Language Models (LLMs)" zur Verbesserung des flüssigen Sprechens und der Satzstruktur in akademischen Vorlesungsaufzeichnungen. Im Zentrum stehen zwei wesentliche Forschungsfragen: die Untersuchung des Einflusses verschiedener "Prompting"-Techniken auf LLMs zur Verbesserung der Vorlesungsaufzeichnungen und die Entwicklung einer Methode zum Trainieren eines Stilklassifikators, der Stil über Inhalt in der Feature-Extraktion für dichte "Embeddings" priorisiert. Mithilfe von Modellen wie Vicuna und LLaMA 2 werden die optimalen Bedingungen für jedes Modell erörtert, wobei die Notwendigkeit individuell angepasster Ansätze bei der Nutzung verschiedener LLMs betont wird. Eine gründliche Analyse zeigte, dass LLaMA 2 in Bezug auf die Stiltransferstärke herausragte, während Vicuna in der Bewahrung des Inhalts überlegen war. Die Few-Shot-Methode, die handgefertigte umformulierte Sätze als Beispiele verwendete, stellte sich als die effektivste Strategie zur Bewahrung des Inhalts heraus. Die Thesis bietet nicht nur einen tiefen Einblick in die aktuellen Technologien, sondern präsentiert auch einen strukturierten Ansatz für das Generierungssystem. Durch die Schließung der bestehenden Lücke in der Integration der Fortschritte im Bereich des natürlichen Sprachverarbeitung in ein Generierungssystem für sprachgewandte Vorlesungsaufzeichnungen, leistet diese Arbeit einen bedeutenden Beitrag zur Realisierung automatisierter Systeme, die in der Lage sind, Vorlesungsaufzeichnungen auf ein Niveau zu transformieren, das mit erfahrenen Rednern vergleichbar ist.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
BERT	Bidirectional Encoder Representations From Transformers
CNN	Convolutional Neural Network
DeBERTa	Decoding-enhanced BERT with Disentangled Attention
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent Unit
KL	Kullback-Leibler
LLaMA	Large Language Model Meta AI
LLM	Large Language Model
LSTM	Long Short-term Memory
ML	Machine Learning
MLP	Multi-layer Perceptron
NLP	Natural Language Processing
RLHF	Reinforcement Learning From Human Feedback
RNN	Recurrent Neural Network
SFT	Supervised Fine-tuning
SGD	Stochastic Gradient Descent
SHAS	Supervised Hybrid Audio Segmentation
T5	Text-to-text Transfer Transformer
TST	Text Style Transfer
TTS	Text-to-speech
WER	Word Error Rate

1. Introduction

Over the last two decades, the educational landscape has undergone substantial transformation. Online learning platforms, once only viewed as a supplemental resource, now play a greater role in ensuring flexibility and accessibility in education. This digital transition, in part driven by technological advances, has lead to the introduction of innovative pedagogical concepts, most notably the hybrid approach in higher education.

In the hybrid model, both traditional classroom instruction and digital tools are integrated, offering a rich and adaptive learning environment. Despite the obvious advantages of additional online resources, educators may find it challenging to maintain a high standard of linguistic fluency and style consistently. Natural interruptions, while often overlooked in live sessions, might become obtrusive in recorded formats. This situation therefore highlights the crucial role of advanced automated tools in refining recordings of lectures, and in doing so ensuring a smooth and effective learning experience, especially in unprecedented situations such as global pandemics.

Consequently, the task at hand, that will be pursued in this thesis, is to analyze a potential implementation of automated systems for enhancing the quality of lecture recordings, thereby helping to augment the learning resources with reduced manual intervention.

1.1. Motivation

The pressing demand for high-quality digital educational resources has never been more pronounced than in the recent times, triggered by the rapid digitalization of learning environments. Lecture recordings stand as a pivotal component in this spectrum, necessitating clarity and coherence to cater to a diverse student populace, including those who are non-native speakers of the instructional language.

Understanding the significance of this venture, it becomes imperative to explore avenues where automation can be a game-changer, reducing the weight on the educators while ensuring an optimal delivery of content. This research arises from the need to foster an educational ecosystem that is efficient, inclusive, and resilient to disruptions.

With advancements in computational linguistics opening up promising pathways, including improved automatic speech recognition (ASR) and increasingly natural text-to-speech (TTS) technologies, there lies a significant opportunity to enhance the quality of lecture recordings, reaching the proficiency of seasoned public speakers.

1.2. Problem Statement

In the realm of digital education, lecture recordings are rapidly becoming indispensable. However, ensuring that these digital mediums are articulate, understandable, and devoid of disfluencies can be an overwhelming task for educators.

This thesis ventures into this territory, aiming to harness the advanced breakthroughs in natural language processing (NLP). While we cannot dispute the power of current technologies, such as state-of-the-art ASR systems including OpenAI's Whisper (Radford et al., 2023), and the evolving capabilities of TTS systems in regard to generating natural sounding speech, a distinct void remains in combining these advances into an end-to-end system for fluent lecture generation.

The void also pertains to the elimination of disfluencies and the application of text style transfer, specifically for spoken content derived from an ASR system. Although the current text style transfer (TST) research landscape is vast, it largely focuses on clear-cut styles like formality or toxicity. Such emphasis does not cater to the specialized needs of improving lecture recordings. Consequently, another exploration of this thesis will be centered on devising comprehensive metrics to evaluate the effectiveness of style transfer.

1.3. Research Questions

In light of the complexities involved in automatically enhancing lecture recordings, this study narrows down its focus to address two critical research questions:

Research Question 1: *How can different prompting techniques influence the effectiveness of large language models (LLMs) in enhancing the fluency and sentence structure of academic lecture recordings, with regards to style transfer strength and content preservation?*

In the subsequent sections, we delve deeper into how LLMs can be used to maintain the essence of the original content while improving linguistic expression, thereby facilitating a richer educational experience.

Research Question 2: *How can a style classifier be trained to prioritize style over content in extracting features for dense embeddings, especially in the presence of skewed content distribution in the training data?*

Further, we explore strategies to steer the focus of classification layers towards style, rather than content, in the face of imbalances in training data, fostering a classifier sensitive to stylistic nuances over mere content representation.

1.4. Thesis Outline

This thesis unveils its narrative in a systematic manner. Commencing with chapter 1, it acquaints readers with the important role of digital learning tools and their inherent challenges. Successively, chapter 2 provides an overview of the vast expanse of foundational concepts, offering an exhaustive review of modern research. This spans areas like ASR, TTS, and TST, setting the stage against which this investigation unfolds.

Progressing further, chapter 3 illuminates the heart of the research. It delineates the experimental framework, discussing everything from data set selection to the technical configurations and assessment metrics. This section provides an in-depth explanation of the strategies devised to address the research questions.

The subsequent chapter 4 then examines the experimental outcomes, presenting a comprehensive analysis that reveals valuable insights and evaluates the proposed methods.

Culminating the thesis, chapter 5 offers a reflective conclusion, encapsulating key discoveries, recognizing potential research constraints, and speculating on future avenues of exploration. This last chapter ensures that readers depart with an encompassing understanding of the research's purpose, methodologies, and wider ramifications.

2. Background & Related Work

In this chapter, we contextualize the research by exploring the underlying concepts and significant contributions to the field. Our exploration begins with the core principles that have shaped contemporary models in a range of current disciplines, detailed in section 2.1. Afterwards, this chapter introduces the three main disciplines of NLP that are involved in our approach on generation of fluent lecture recordings.

2.1. Fundamentals

Here we outline the elementary framework of neural networks and deep learning architectures, establishing a ground knowledge that will facilitate a deeper understanding of the advanced topics to follow. Then, we cover concepts that have led to significant advancements in the field, such as attention mechanisms, the transformer architecture, the concept of LLMs and also techniques that are necessary for almost all tasks involving NLP, such as tokenization.

2.1.1. Artificial Neural Networks

Artificial neural networks (ANNs) are a category of machine learning (ML) models that draw inspiration from the biological neural networks found in animal brains. The first, relatively simple, neuron model aimed at imitating real neural structures was introduced 80 years ago, the so-called McCulloch-Pitts neuron (McCulloch and Pitts, 1943). Since then, much research has gone into further developing and refining those early approaches, and in many disciplines today, ANNs are the foundation of many, if not most, state-of-the-art models.

In today's form, ANNs can be considered universal function approximators. According to one of the universal approximation theorems, they "are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available" (Hornik et al., 1989). The concrete mathematical function that the ANN ultimately represents is determined by a set of learnable parameters within the model.

The functionality of ANNs can best be elaborated on by looking at the smallest building block of any network, the perceptron. A perceptron takes an input vector $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ of values and transforms it into a single value $\mathbf{y} \in \mathbb{R}$. It does this by first calculating the weighted sum of the input values, using a weight vector $\mathbf{w} = (w_1, w_2, ..., w_n) \in \mathbb{R}^n$ and additionally adding a bias $b \in \mathbb{R}$. The result of this is then put into an activation function σ to obtain $\mathbf{y}$.

$$\mathbf{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

The activation function is usually some nonlinear function that ensures $\mathbf{y} \in (-1, 1)$, $\mathbf{y} \in (0, 1)$ or $\mathbf{y} \in [0, \inf)$, depending on the specific activation function being chosen. By using the Heaviside step function H with

$$H(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

as the activation function, the perceptron can also be seen as a binary linear classifier, which describes a $(n-1)$-dimensional hyperplane that separates the input space into two classes. A single perceptron is not powerful enough to learn most functions though, which can easily be seen by observing that it is impossible for a perceptron to learn the simple XOR function. To overcome this, ANNs combine these building blocks into multi-layer perceptrons (MLPs).

MLPs build upon the foundation laid by the single-layer perceptron by stacking several layers of neurons (or nodes) to form a deep network. This architecture consists of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple nodes that are fully connected to the nodes in the adjacent layers, facilitating complex function approximations.

To further expand on the computation process, a MLP with one hidden layer will be considered in the following. The output of the neurons in the hidden layer can be represented as a vector $\mathbf{h}$ where each entry h_i is computed as:

$$h_i = \sigma\left(b_i^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right)$$

Here, $b_i^{(1)}$ and $w_{ij}^{(1)}$ are the bias and weight associated with the i-th neuron in the hidden layer, respectively. The superscript (1) denotes the layer number.

To leverage matrix operations for efficient computation, we can rewrite the above equation in matrix form:

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

where:

- $\mathbf{W}^{(1)}$ is the weight matrix of the first hidden layer with dimensions $(m \times n)$, where m is the number of neurons in the hidden layer and n is the number of input values.
- $\mathbf{x}$ is the input vector with dimensions $(n \times 1)$.

- $\mathbf{b}^{(1)}$ is the bias vector of the first hidden layer with dimensions $(m \times 1)$.
- σ is applied element-wise to the resulting vector.

Following this, the output of the network can be computed as:

$$\mathbf{y} = \sigma(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

where:

- $\mathbf{W}^{(2)}$ is the weight matrix of the output layer with dimensions $(p \times m)$, where p is the number of neurons in the output layer.
- $\mathbf{h}$ is the output vector from the hidden layer with dimensions $(m \times 1)$.
- $\mathbf{b}^{(2)}$ is the bias vector of the output layer with dimensions $(p \times 1)$.

Through the stacking of multiple layers and the nonlinear transformations induced by the activation functions at each layer, MLPs can learn to approximate complex, nonlinear functions to a high degree of accuracy.

2.1.2. Training

Training neural networks like MLPs involves iteratively tuning the weights and biases of the network to minimize a loss function that measures the discrepancy between the predictions made by the network and the actual target values in the dataset. This section will illuminate two pivotal concepts in the training process: gradient descent and backpropagation.

Backpropagation is a technique to efficiently compute the gradient of the loss function with respect to each weight and bias in the network, utilizing the chain rule of calculus. Starting from the output layer and moving backward through each layer, it computes the gradient step-by-step, which will be utilized in the gradient descent algorithm to update the weights and biases.

Gradient descent is an optimization algorithm that iteratively updates the weights and biases of the network to find the minimum of the loss function. This is achieved by moving in the direction of the steepest descent, i.e., the negative of the gradient. The rules for updates are as follows:

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \cdot \frac{\partial L}{\partial w_{ij}^{(l)}}$$

$$b_{i}^{(l)} := b_{i}^{(l)} - \eta \cdot \frac{\partial L}{\partial b_{i}^{(l)}}$$

where:

- η is the learning rate, a hyperparameter that influences the step size during the optimization process.
- $w_{ij}^{(l)}$ and $b_i^{(l)}$ are the weight and bias for the connection from the j-th neuron in layer $l-1$ to the i-th neuron in layer l.

The learning rate η must be chosen judiciously to ensure convergence towards the minimum without oscillations or divergence. Furthermore, several variants of the gradient descent, such as stochastic gradient descent (SGD) (Kiefer and Wolfowitz, 1952) and mini-batch gradient descent, have been developed to enhance convergence speed and stability.

Through gradient descent, aided by backpropagation for efficient gradient computation, ANNs can learn to accurately approximate complex functions, evolving from a network with random initial parameters to a potent tool in various machine learning applications.

2.1.3. Encoder-Decoder Architecture

A cornerstone in modern deep learning architectures, especially in the context of sequence-to-sequence prediction tasks, is the encoder-decoder framework (Cho et al., 2014). This paradigm essentially divides the model into two distinct yet cooperative segments: the encoder and the decoder.

The **encoder** takes the input data and transforms it into a higher-level representation, generally a dense latent space that encapsulates the critical attributes and patterns in the data, thereby forming an "understanding" or "abstraction" of the input. This representation is often denoted as the context vector, which serves as the bridge conveying the essential information from the encoder to the decoder. This part of the network learns to discern and condense the underlying structures and crucial components from the inputs, serving as a knowledgeable interpreter of the given data. A mathematical representation of the encoder can be given as:

$$\mathbf{c} = f(\mathbf{x}; \theta_{\text{enc}})$$

where:

- $\mathbf{c}$ is the context vector created by the encoder.
- f is the function represented by the encoder network.
- $\mathbf{x}$ is the input data.
- θ_{enc} are the parameters of the encoder.

Following this, the **decoder** receives the context vector and works to reconstruct or translate the original input data into the desired output, leveraging the high-level representation garnered from the encoder. The decoder effectively learns to build a mapping from the abstract representation to actual outputs that align with the target domain's structure and intricacies. It can be mathematically represented as:

$$\mathbf{y} = g(\mathbf{c}; \theta_{\text{dec}})$$

where:

- $\mathbf{y}$ is the output data.
- g is the function represented by the decoder network.
- $\mathbf{c}$ is the context vector received from the encoder.
- θ_{dec} are the parameters of the decoder.

The cooperative operation of the encoder and decoder through this architectural principle facilitates a more profound understanding and handling of complex data structures, a trait especially vital in handling tasks such as machine translation, image captioning, and various NLP undertakings.

Moreover, this conceptual framework lays a solid ground for introducing transformer architectures, which further extends and sophisticates the encoder-decoder principle by introducing mechanisms such as attention, which allows the network to focus on different parts of the input for different tasks, paving the way for highly flexible and potent models in the deep learning landscape.

2.1.4. Attention Mechanism

The attention mechanism, a cornerstone in the field of deep learning, has notably expanded the capabilities of numerous neural network architectures including the long short-term memory (LSTM) and the gated recurrent unit (GRU). Initially designed to augment sequence-to-sequence models in machine translation systems (Bahdanau et al., 2014), it has become a core component in a plethora of state-of-the-art models, enhancing performance in tasks ranging from natural language processing to computer vision.

The fundamental concept behind the attention mechanism is the dynamic weighting of the importance of different parts of the input when computing the output, effectively allowing the network to "attend" to different parts of the input for various tasks. This selective focus facilitates a more nuanced understanding and representation of input data.

In its initial incarnations, attention was applied in sequence-to-sequence models, assisting the decoder in focusing on different segments of the input sequence when generating each word in the output sequence, a process that emulates dynamic alignment of inputs and outputs during training. However, the true potential of attention mechanisms was fully realized with the introduction of self-attention (Cheng et al., 2016), where relationships between different parts of the input are leveraged to compute the representation at each position in the input sequence.

A widely adopted formulation of attention, especially in transformer models (Vaswani et al., 2017), which we delve into in the following section, is defined as:

$$\mathbf{a}_t = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

where:

- $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value matrices, derived from the input.
- d_k is the dimensionality of the queries and keys.
- The softmax function ensures the weights are normalized.

Here, the dot product between the query and key vectors is divided by the square root of the dimension of those vectors in order to ensure that high dimensionality of the vectors does not lead to exploding or vanishing gradients in training, which can be caused by disproportionately large dot product results.

In the next step, the context vectors are derived as a weighted sum of the value vectors, guided by the attention scores computed from the query and key vectors:

$$\mathbf{c}_t = \sum_{i=1}^{T} a_{ti}\mathbf{v}_i$$

where:

- $\mathbf{c}_t$ is the context vector at time t.
- T is the length of the input sequence.
- a_{ti} represents the attention weight for the i-th input at time t.
- $\mathbf{v}_i$ is the value vector corresponding to the i-th input.

The self-attention mechanism forms the backbone of the transformer architecture, facilitating the parallelization of computation and thus enabling the processing of longer sequences compared to older architectures based on recurrent or convolutional layers. It fundamentally shifted the approach towards sequence modeling, paving the way for more intelligent and adaptable neural network architectures.

2.1.5. Transformer Architecture

In recent years, the transformer architecture has emerged as a dominating approach in the field of deep learning, particularly within natural language processing tasks (Vaswani et al., 2017). The architecture, fundamentally rooted in the encoder-decoder framework, augments this structure with self-attention mechanisms, which enables the model to weigh the influence of each word (or sub-word) in a sequence when predicting an output.

One of the central advantages of transformers over previous sequence-to-sequence models, such as plain recurrent neural networks (RNNs) or LSTMs, is their ability to handle long-range dependencies in a sequence more effectively. The self-attention mechanism allows each word in the input sequence to focus on every other word, thereby capturing intricate patterns and relationships that might be missed by models that process sequences step-by-step. Furthermore, transformers facilitate parallel computation across sequence positions, which significantly speeds up training compared to plain RNNs and LSTMs, which require sequential computation. This increased efficiency in training not only speeds up the learning process but also allows for the construction of deeper models, capable of learning more complex patterns and hierarchical representations. This attribute, coupled with the elimination of the necessity to learn sequential transition dynamics, often results in transformers outperforming other architectures in a wide array of NLP tasks.

Architecture Overview

The original transformer consists of an **encoder** and a **decoder**, each comprising multiple identical layers stacked atop one another, thereby facilitating deep representation learning. The fundamental unit of both the encoder and decoder is the self-attention mechanism, supplemented with feedforward neural networks. A detailed depiction of the architecture is presented in Figure 2.1.

The **encoder** is composed of a stack of N identical layers. Each layer contains two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward neural network. A residual connection surrounds each of the two sub-layers, followed by layer normalization. Thus, the output of each sub-layer is $\text{LayerNorm}(x + \text{SubLayer}(x))$, where SubLayer(x) is the function implemented by the sub-layer itself. The encoder accepts a sequence of token embeddings combined with positional encodings as its input.

The **decoder** also comprises N identical layers, but includes an additional sub-layer that performs multi-head attention over the encoder's output. This sub-layer is inserted between the self-attention layer and the feedforward neural network. Similar to the encoder, each sub-layer in the decoder is followed by a residual connection and layer normalization.

Multi-Head Attention

Central to the transformer is the **multi-head attention** mechanism, which allows the model to focus on different parts of the input for different representational subspaces. This mechanism employs several independent self-attention layers (the "heads") running

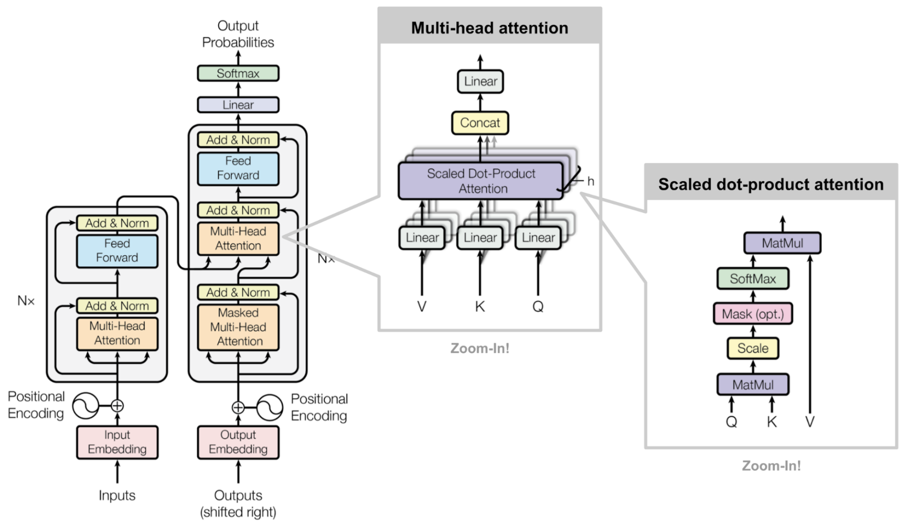

Figure 2.1.: Original transformer architecture in paper by Vaswani et al. (2017) with multi-head attention and scaled dot-product attention.

in parallel, each working on different projections of the input. Formally, the multi-head attention can be defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each head_i is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here, W_i^Q, W_i^K, W_i^V, and W^O are learnable parameter matrices, and the Attention function is as defined in the initial description of the self-attention mechanism.

Positional Encoding

Since the transformer does not inherently understand the order of tokens in a sequence, **positional encodings** are added to the embeddings of the tokens to grant the model positional information. These encodings are designed to be added to the embeddings, thereby facilitating the learning of the positional relationships between tokens. The positional encodings can be defined using various functions; a popular choice, as defined in the original transformer paper, uses sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

where *pos* represents the position of the token in the sequence, i refers to the dimension index, and d_{model} is the dimensionality of the model.

Feedforward Neural Networks

Each layer of the transformer contains a **feedforward neural network** that operates independently on each position. Comprising two linear transformations with a ReLU activation function in the middle, it can be mathematically represented as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where W_1, W_2, b_1, and b_2 are learnable parameters, and x is the input to the feedforward neural network.

Applications and Extensions

Since its inception, the transformer architecture has been foundational to several breakthroughs in natural language processing. Models such as bidirectional encoder representations from transformers (BERT) (Devlin et al., 2019), the generative pre-trained transformer (GPT) (Radford et al., 2018), and the text-to-text transfer transformer (T5) (Raffel et al., 2020) have set new performance benchmarks in numerous NLP tasks. Moreover, transformers have been adapted and extended to various other domains, including image and video processing, demonstrating its versatile applicability and profound impact on the field of deep learning.

2.1.6. Generative Models

Generative models have received an increasing amount of attention in recent years and there have been several breakthroughs in the field, especially in NLP with the rise of LLMs and in computer vision with the introduction of diffusion models (Ho et al., 2020). This category of models generally focuses on generating new data points that are similar to other instances in a given dataset. They achieve this, as most ANNs do, by trying to understand the underlying patterns and distributions that the data is built on.

Contrary to purely discriminative models, which aim at learning the conditional probability distribution $P(X|Y)$ of the data X and the labels Y in order to determine the boundaries between different classes in the data, generative models try to learn the joint distribution $P(X, Y)$, or $P(X)$, if no labels are available. This then allows for a broader variety of applications made possible by generative models, such as generating new data points by sampling from the distribution $P(X|Y)$ for disciplines like data augmentation, super-resolution and synthetic data generation. Additionally, it is important to note that generative models are always able to be used for discriminative tasks as well, simply by computing $P(Y|X)$ using Bayes' theorem.

Considering the preceding subsection 2.1.5, it is vital to discuss the role of the transformer architecture in catapulting the capabilities of generative models, especially in NLP, to unprecedented heights. One of the key concepts is the encoder-decoder paradigm mentioned in subsection 2.1.3. In encoder-decoder transformers, the encoder part processes the input to form a contextualized representation in the form of a dense vector. The decoder then uses this representation to autoregressively generate the output. The autoregressive generation strategy makes sure that it is conditioned on the previous outputs at each step, therefore providing outputs with coherent and logical sequences, which is particularly essential for natural language generation. This is why encoder-decoder transformers have been widely used in current neural machine translation systems and outperformed previous methods based on recurrent neural networks (Lakew et al., 2018). Here, by using the dense representations of the source language input tokens (see section 2.1.7 for explanation of the concept of tokenization) from the encoder, the decoder autoregressively generates the translation in the target language.

Another approach to autoregressive data generation are decoder-only transformers. These models implement a slightly different approach, since they are utilizing a powerful decoder to generate the output step by step, while taking into account all the previous outputs in the sequence. The most relevant examples for this strategy in recent times have been the GPT models, where enormous potential has been shown in being able to generate human-like text based on the preceding context.

2.1.7. Tokenization

In the context of NLP, tokenization is a fundamental step that involves breaking down a large paragraph into sentences or words, essentially into smaller pieces referred to as tokens (Jurafsky and Martin, 2023). In the context of ANNs, these tokens are essential for enabling them to understand the semantics of text. Depeding on the needed granularity of the task, tokenization can be performed at different levels, such as word, sub-word, or even character level. In the succeeding paragraphs, the essence and different approaches to tokenization will be discussed extensively.

Tokenization is essential as it directly affects the quality of the representation that can be learned from the text. Precise tokenization mechanisms help in preserving the semantics, which is crucial for a structured and insightful analysis of the text. Additionally, it can assist in removing irrelevant words or characters from the text, such as spaces in the case of word-based tokenization, helping with more efficient data processing.

Historically, a range of techniques has been developed for tokenization, broadly categorized into the following:

- **Word-based Tokenization**: In this approach, the text is broken down into individual words. This is generally the most straightforward type of tokenization and is usually achieved using simple delimiters such as spaces or punctuation marks.
- **Sub-word Tokenization**: Recognizing that words can often contain meaningful sub-components, this approach further breaks down words into smaller units. Techniques such as Byte-Pair Encoding (BPE) (Sennrich et al., 2016), Unigram Language Model

(Kudo, 2018), and SentencePiece (Kudo and Richardson, 2018) are popular in this category. Sub-word tokenization is particularly useful as it helps with handling rare and out-of-vocabulary words, which might pose a problem for simple word-based tokenization.

- **Character-based Tokenization**: This strategy involves breaking down text into individual characters. While it might help in learning very fine-grained patterns, it usually results in much longer sequences, which in turn increases the computational requirements.

In the context of transformer models, tokenization plays a pivotal role, as it directly influences the effectiveness of the self-attention mechanism, which, as outlined in the preceding sections, leverages token-level representations to understand the semantic relationships between different parts of the text. It is therefore imperative to choose a tokenization strategy that aligns with the complexity and the nuances of the task at hand.

As referenced in the previous subsection 2.1.6, tokenization forms the basis of understanding how encoder and decoder mechanisms perceive and generate textual data. Choosing a particular tokenization strategy can notably influence the performance of transformer models in various NLP tasks, which further highlights its critical role in the development of advanced NLP systems.

2.1.8. Large Language Models

In recent years, the field of NLP has witnessed substantial advancements owing to the development of LLMs, a category of deep learning models characterized by their substantial parameter scale and ability to generate highly coherent and contextually relevant text based on expansive training on diverse textual corpora (Brown et al., 2020). The pre-eminent approach to training these models embodies a two-step procedure comprising pre-training and fine-tuning phases.

During the pre-training phase, these models are exposed to enormous amounts of text data, enabling them to learn linguistic patterns, grammatical rules, and even acquire substantial factual knowledge through self-supervised learning methodologies. The fine-tuning phase follows, wherein supervised fine-tuning (SFT) aids in the specialization of the models for a variety of specific tasks by leveraging a narrower dataset which is task-specific, facilitating refined performance on particular tasks, an approach grounded in supervised learning paradigms (Howard and Ruder, 2018).

Recent advancements have seen the scaling up of LLMs to unprecedented levels, with models such as GPT-3 boasting as many as 175 billion parameters, a tenfold increase compared to its predecessors. This scaling has been pivotal in enhancing the task-agnostic, few-shot performance of these models, sometimes even rivaling the performance of fine-tuned approaches (Brown et al., 2020). Remarkably, GPT-3 achieves this performance without any gradient updates or fine-tuning, relying solely on text interactions for task and few-shot demonstrations. However, it is pertinent to note that while zero-shot performance improves with model size, few-shot performance escalates more rapidly, indicating a higher proficiency in in-context learning with larger models. This phenomenon suggests that LLMs can be perceived as meta-learners, integrating slow outer-loop gradient descent learning with rapid in-context learning facilitated through the model's context activations, a promising avenue for achieving state-of-the-art performance in various NLP tasks (Brown et al., 2020).

An emergent challenge in the utilization of LLMs is the necessity for mechanisms to prevent the generation of inappropriate or biased content. The evolving strategy to address this issue involves reinforcement learning from human feedback (RLHF), a technique through which the models are iteratively fine-tuned based on evaluative feedback from human reviewers, engendering a more controlled and ethically aligned output (Bai et al., 2022).

As we stand at the beginning of an era dominated by LLM-enabled applications ranging from drafting articles to facilitating sophisticated dialogue systems, it is essential for the research community to pay special attention to accompanying societal challenges and issues. It necessitates a rigorous inspection of the ethical dimensions surrounding the deployment of LLMs, safeguarding against misuse while simultaneously cultivating an environment of responsible artificial intelligence (AI) complying with human values and norms.

2.2. Automatic Speech Recognition

ASR, fundamentally, is the technology that converts spoken language into written text. This field has been central in facilitating human-computer interaction, thereby finding applications in a myriad of domains, especially on mobile devices, including transcription services, voice assistants, and many others (Yu and Deng, 2014).

ASR is grounded in deep learning and NLP, leveraging complex approaches to understand and transcribe spoken language. The process generally involves several stages including feature extraction, acoustic modeling, and language modeling, each contributing to the accurate transcription of speech into text.

The beginning of ASR can be traced back to the 1950s at Bell Labs, with IBM's Shoebox (1961) being one of the first machines capable of recognizing spoken digits. Over the decades, the field has evolved exponentially, with notable milestones including the introduction of Hidden Markov Models in the 1980s, and the advent of deep learning approaches in recent years, which have significantly enhanced the accuracy and efficiency of ASR systems (Juang and Rabiner, 2005).

Recent developments in the field have been characterized by the integration of transformer architectures, as discussed in subsection 2.1.5, which have enabled the learning of more complex patterns and relationships in spoken language. Moreover, end-to-end ASR systems have emerged as a dominant approach, simplifying the traditional ASR pipeline and offering improvements in both accuracy and efficiency (Chan et al., 2016).

Evaluating ASR systems traditionally involves the use of word error rate (WER), a metric that quantifies the number of substitutions, deletions, and insertions needed to match the transcribed text with the reference text (Klakow and Peters, 2002). More recently, researchers have been exploring metrics that go beyond syntactic accuracy to consider semantic correctness, evaluating the preservation of meaning and intent in the transcribed text. This is in line with the broader trends in NLP, emphasizing understanding and generation of language that is not just technically correct but also contextually and semantically aligned with human communication norms (Rugayan et al., 2022; Roux et al., 2022).

2.3. Text Style Transfer

TST, a sub-discipline within NLP, aims to alter the stylistic properties of a text while preserving its core content and semantic essence (Jin et al., 2022). This task is often formulated as a sequence-to-sequence learning problem where models are trained to map input sentences to output sentences that convey the same content but with altered stylistic attributes.

Historically, the text style transfer can trace its roots to the early works in machine translation and paraphrase generation. The early approaches often relied on rule-based systems and manually crafted features. However, with the rising popularity of deep learning and particularly the transformer architectures, the field has witnessed a paradigm shift, moving towards data-driven approaches that leverage the potential of large annotated corpora to learn the nuanced mappings between different styles automatically (Li et al., 2018; Fu et al., 2018).

The emergence and rise of LLMs, as presented in subsection 2.1.8, have further pushed the boundaries in the text style transfer domain. Recent works have extensively leveraged pre-trained LLMs for style transfer tasks, fine-tuning them on specific style transfer datasets to achieve state-of-the-art performance (Krishna et al., 2020). The sheer scale and learning capabilities of these models facilitate a deep understanding of linguistic nuances, enabling more fluent and natural style transfer outputs.

In this context, most efforts in the discipline have been centered around attributes such as sentiment, formality, and toxicity. However, the current literature has not extensively explored the framing of disfluency removal and sentence structure improvement as a style transfer task, which this thesis intends to do.

Evaluating the performance of text style transfer models presents a multifaceted challenge. Recent evaluation metrics include learned evaluation metrics like BLEURT (Sellam et al., 2020) for fluency and content preservation, and style-specific metrics to assess the degree of style transfer. Moreover, manual evaluations involving human annotators are often conducted to judge the quality of the generated texts from various aspects including grammaticality, preservation of meaning, and the effectiveness of style transfer (Mir et al., 2019).

TST stands as a vibrant and dynamically evolving field, holding the promise of enhancing various applications including highly customizable content generation, data augmentation, and the personalization of conversational agents, among others.

2.4. Text-to-speech

The field of TTS is essential to the development and enhancement of human-computer interaction, focusing on the conversion of written text into spoken language. This multidisciplinary field navigates through linguistics, computer science, and acoustic engineering to develop systems capable of generating natural and coherent speech, facilitating a more accessible and interactive technological landscape.

Looking back, the evolution of TTS can be traced back to the early 1960s with the development of systems that could synthesize speech from text using rudimentary rule-based approaches. Progress in the field was steady, with significant milestones such as the introduction of formant synthesis, followed by the upcoming of concatenative synthesis, which leveraged recorded speech samples to generate speech signals, thus allowing for a more natural speech output.

Recent advancements in TTS have been characterized by the rise of deep learning methodologies, which facilitated the transition from concatenative synthesis to parametric synthesis approaches such as statistical parametric speech synthesis (Zen et al., 2009). This evolution reached the current state of the art with the development of end-to-end neural network approaches, such as WaveNet (Oord et al., 2016) as one of the first ones, which drastically enhanced the quality of synthesized speech, moving closer to a truly human-like speech synthesis.

A significant breakthrough in recent years is the introduction of unsupervised learning of audio representations, a methodology that leverages large amounts of unlabeled audio data to learn rich audio representations that can be utilized in various downstream tasks. Examples of this approach include the development of models such as AudioLM (Borsos et al., 2023) and Bark, which have opened new avenues in the audio processing field, enhancing the capabilities of TTS systems and offering potential improvements in voice synthesis and other related applications.

Evaluation of TTS systems historically relied on objective metrics such as Mel Cepstral Distortion. However, with the intricate nuances involved in human speech, the importance of subjective evaluations has risen, wherein human listeners rate the naturalness, intelligibility, and overall quality of the synthesized speech, providing a more holistic assessment of the system's capabilities (Hinterleitner, 2017). Later on, metrics such as MOS (Mean Opinion Score) have been adopted, offering a standardized evaluation strategy that incorporates both objective and subjective measures (Viswanathan and Viswanathan, 2005).

In line with the current trajectory, the field of TTS is at the forefront of many exciting possibilities, with ongoing research focused on perfecting the nuances of synthesized speech to be indistinguishable from human speech, further exploring the realms of emotional and expressive speech synthesis, and continuously striving for a more inclusive and accessible digital ecosystem.

3. Approach

The following chapter will give light to the approach that the following parts of this thesis will revolve around. First, it introduces our suggested structure for an automated generation system for fluent lecture recordings. After that, the models that are part of the system are briefly presented and explained in more detail. Furthermore, it presents the data being used for our approach and how we train classifiers with them for evaluating the style transfer strength. Finally, the employed evaluation metrics for the TST task will be presented.

3.1. Structure of the Generation System

The generation system envisioned in this research is a tripartite structure, designed carefully to bring about fluency and coherence in lecture recordings. This section outlines each component and expands on the rationale behind the choices made in system configuration.

Transcription

The initial phase in the generation system is anchored by a high-caliber ASR system, which transcribes the audio input from the original lecture recordings meticulously. Our choice for this role is OpenAI's Whisper, revered for its state-of-the-art transcription quality, complemented by its adeptness in bypassing minor disfluencies inherently present in verbal communications.

To further enhance the transcription accuracy, we employ supervised hybrid audio segmentation (SHAS) to dissect the original audio file into manageable segments. These fragments are then individually transcribed using Whisper, and subsequently combined to constitute a comprehensive transcript. Although OpenAI's implementation of Whisper is already capable of handling elongated sequences, there still are some problems with directly using it without prior segmentation, which will later be discussed in subsection 3.2.2.

We consciously abstain from an in-depth evaluation of this segment in our thesis, considering the rarely matched precision of Whisper and the absence of a golden standard transcription for the specific lecture recordings at our disposal.

Rephrasing

Progressing to the second tier of our system, we delve into the realm of TST, utilizing LLMs to enhance the textual output derived from the ASR system. The objective here is to refine the sentence structure while concurrently eliminating disfluencies. At the same time, the rephrased sentence should still be semantically equivalent to the original.

This venture entails the deployment of different prompting techniques with LLMs to rephrase each sentence sequentially, striving for linguistic elegance and fluency. An illustrative list of varied prompting methods and prompts are listed for reference in the appendix A.3.

Our research undertakes a comparative analysis of two formidable LLMs — Vicuna, a fine-tuned iteration of the original LLaMA model, and its successor, LLaMA 2, which has been optimized for chat applications leveraging RLHF. These models are orchestrated to achieve the overarching goal of linguistic refinement with an emphasis on content preservation.

Speech Generation

Concluding the structure is the TTS system, the final yet crucial stage that breathes life into the refined transcripts, transforming them into fluent lecture recordings. We have zeroed in on the Bark model developed by Suno AI for this important role, primarily for its additional proficiency in voice cloning, thereby enabling the generation of recordings congruent with the original lecturer's voice timbre.

This endeavor mandates prior authorization to ethically align with voice cloning guidelines, ensuring a moral adherence to personal privacy. Similar to the ASR system, this component will remain exempt from further evaluations, directing the focal point of the research towards the effectiveness of the style transformation.

Video Generation

In order to truly be able to generate a fluent version of an existing lecture recording, one would further need to detect slide transitions in the recorded slides and then match the newly generated sentences to the slide that they were originally communicated at. An ML approach for detecting these lecture slide transitions would be the SliTraNet model, as described by Sindel et al. (2022). Just like before, this thesis will not further attend to this particular part of the system in later chapters and it is only mentioned here for reasons of completeness.

3.2. Models

The next subsections will cover all the models being used in the generation system. They will highlight each of their structures, give a general overview on how they work, how they were trained and discuss their usage in the overarching system.

3.2.1. SHAS

The SHAS model stands as a essential component in the automated generation system, primarily focusing on the optimal segmentation of long audio recordings, a task that has primarily been addressed for achieving high-quality speech translation, but can also be very useful for our speech transcription task. The arisen necessity to combine the SHAS segmenter with the Whisper ASR model will be addressed in subsection 3.2.2.

This subsection primarily outlines the architecture and functioning of the SHAS model, in general pronouncing its role in enhancing the efficiency and accuracy of speech translation systems.

The architecture of SHAS is grounded on the utilization of a pre-trained wav2vec 2.0 model (Baevski et al., 2020) to derive speech representations, which are then employed to train a classifier. This classifier is tasked with identifying the optimal frames for segmenting the audio. The segmentation process is guided by a probabilistic Divide-and-Conquer algorithm that iteratively splits the audio at the frame with the lowest probability, ensuring that all segments remain below a predetermined length (Tsiamas et al., 2022).

Training the classifier involves leveraging manually segmented speech corpora to learn the optimal segmentation points. The wav2vec 2.0 model plays a crucial role here, providing the necessary speech representations that facilitate the training process. The classifier is trained to recognize the frames that should be included in a segment, thereby learning to mimic the manual segmentation process effectively.

Empirical evaluations have demonstrated the efficacy of the SHAS model, showcasing its ability to retain 95-98% of the BLEU score achieved through manual segmentation. Furthermore, it exhibited high zero-shot performance on unseen languages, indicating its robustness and adaptability to different linguistic contexts without necessitating training on them. This aspect of SHAS stands as a testament to its generalizability and its potential to be a versatile tool in speech translation systems (Tsiamas et al., 2022).

In the context of the automated generation system proposed in this thesis, the SHAS model serves as an important element in processing long audio recordings. By ensuring optimal segmentation, it substantially supports the subsequent stage of speech recognition. Its integration promises a system that is not only efficient but also maintains a high standard of transcription quality, bringing us a step closer to the goal of transforming lecture recordings.

In conclusion, the SHAS model emerges as a sophisticated tool originally from the landscape of speech translation, offering a solution to the longstanding issue of audio segmentation. Its architecture, grounded on the wav2vec 2.0 model, and its proven performance in empirical evaluations, position it as a promising component in the development of advanced NLP systems.

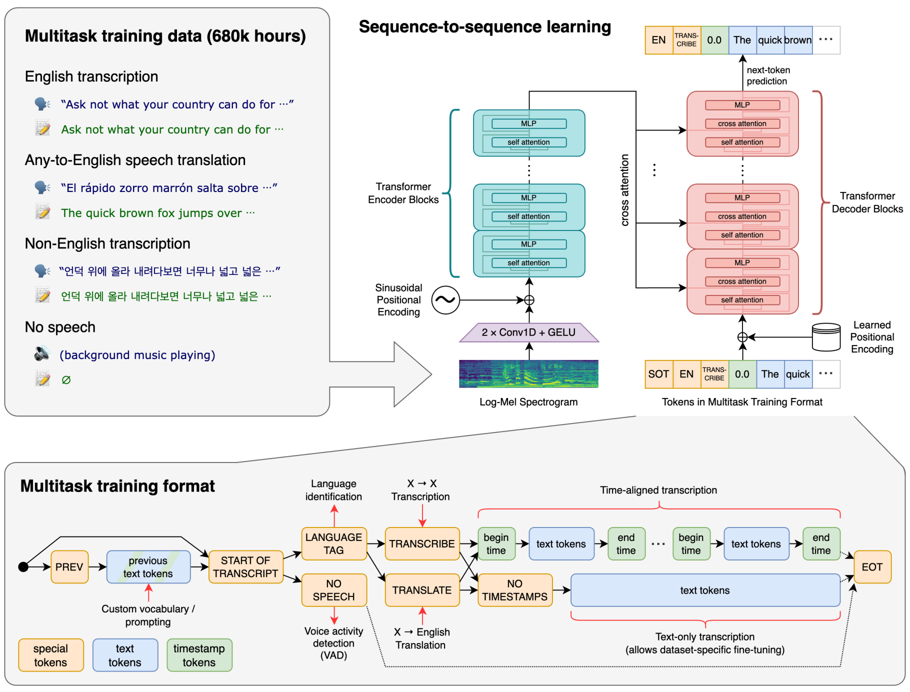
Multitask training data (680k hours)
English transcription
"Ask not what your country can do for ⋯"
Ask not what your country can do for ⋯
Any-to-English speech translation
"El rápido zorro marrón salta sobre ⋯"
The quick brown fox jumps over ⋯
Non-English transcription
"언덕 위에 올라 내려다보면 너무나 넓고 넓은 ⋯"
언덕 위에 올라 내려다보면 너무나 넓고 넓은 ⋯
No speech
(background music playing)
∅
Sequence-to-sequence learning
Transformer Encoder Blocks
Transformer Decoder Blocks
MLP
self attention
cross attention
Sinusoidal Positional Encoding
Learned Positional Encoding
2 × Conv1D + GELU
Log-Mel Spectrogram
Tokens in Multitask Training Format
next-token prediction
EN TRANSCRIBE 0.0 The quick brown ⋯
SOT EN TRANSCRIBE 0.0 The quick ⋯
Multitask training format
Language identification
X → X Transcription
Time-aligned transcription
PREV
previous text tokens
START OF TRANSCRIPT
LANGUAGE TAG
NO SPEECH
TRANSCRIBE
TRANSLATE
begin time
text tokens
end time
NO TIMESTAMPS
EOT
Custom vocabulary / prompting
Voice activity detection (VAD)
X → English Translation
Text-only transcription (allows dataset-specific fine-tuning)
special tokens
text tokens
timestamp tokens

Figure 3.1.: Whisper architecture as in Radford et al., 2023.

3.2.2. Whisper

The Whisper ASR system is an essential element in the automated generation system, working in tandem with the SHAS model to facilitate the transcription of segmented audio recordings. This subsection delves into the architecture and functionalities of the Whisper model, highlighting its role in enhancing the quality and effectiveness of speech transcription.

Whisper is an ASR system developed by OpenAI. It is trained on 680,000 hours of multilingual and multitask supervised data collected from the web. The architecture of Whisper is built upon a deep learning framework, utilizing convolutional neural networks (CNNs) to process audio signals and transcribe them into text. The detailed architecture is illustrated in figure 3.1, as referenced in Radford et al., 2023.

The integration of Whisper into the automated generation system is envisioned to streamline the transcription process, offering high precision in converting audio recordings into textual data. Its collaboration with the SHAS model ensures that the segmented audio is transcribed optimally, better preserving the nuances and intricacies of the original speech than with using whisper alone. Not only that, it also prevents Whisper from getting stuck in a sort of "no-punctuation mode", where the model, due to its autoregressive nature, only transcribes the words being spoken in lowercase letters, but does not add any punctuation at all. The latter has been experienced multiple times during early experiments, especially when transcribing recordings with unclear sentence structure, rendering part of the generated transcription unparsable. Consequentially, this further signifies the need for combining the Whisper model with a segmenter in order to gain accurate, parsable transcriptions for later steps.

Empirical evaluations of the Whisper model have underscored its robust performance in speech recognition tasks. Its training on a vast corpus of multilingual data equips it with the ability to handle a wide array of linguistic contexts, thereby promising versatility and adaptability in its operations.

In the broader perspective of this thesis, the Whisper model stands as a cornerstone in the realization of fluent lecture recordings. It promises to enhance the accuracy and fluency of transcriptions, thereby facilitating a smoother transition to subsequent processes such as style transfer. Its role is thus central to achieving the overarching goal of this thesis, which is to develop an automated system capable of generating fluent lecture recordings with high fidelity to the original content.

In conclusion, the Whisper ASR system emerges as a powerful tool in the automated generation system, bringing to the table its advanced architecture and proven efficacy in speech recognition tasks. Its integration promises to elevate the quality of transcriptions, paving the way for fluent and coherent lecture recordings that retain the essence of the original speech (Radford et al., 2023).

3.2.3. Vicuna

The Vicuna model, specifically the Vicuna-13B variant, is an open-source chatbot developed by the Vicuna team, and it represents a significant stride in the chatbot landscape. It was fine-tuned using the LLaMA base model on a dataset comprising around 70,000 user-shared conversations harvested from ShareGPT.com. This fine-tuning has endowed Vicuna with the ability to generate more detailed and well-structured responses compared to other models such as Alpaca and even matches the quality of responses generated by ChatGPT in certain aspects (Chiang et al., 2023).

The Vicuna-13B leverages an enhanced dataset and a scalable infrastructure, which is a testament to the rapid advancements in LLMs. The training involved fine-tuning a LLaMA base model with the conversations collected, and enhancing training scripts initially provided by the Alpaca project to better handle multi-turn conversations and long sequences. The training utilized PyTorch FSDP on 8 A100 GPUs, completing in a single day. Noteworthy is the increase of the maximum context length to 2048, facilitated through memory optimizations such as gradient checkpointing and flash attention, which significantly increased the GPU memory requirements. The training also saw a substantial cost reduction, achieved through the use of SkyPilot managed spot instances, bringing down the cost to around $300 for the 13B model.

Preliminary evaluations, albeit non-scientific and requiring further rigorous analysis, indicate that Vicuna-13B achieves over 90% of the quality exhibited by OpenAI's ChatGPT and Google Bard, outperforming other models like LLaMA and Stanford Alpaca in over 90% of the cases. The evaluations utilized GPT-4 as a judge, leveraging its ability to produce consistent ranks and detailed assessments of chatbot responses. This innovative evaluation framework, still in its nascent stage, presents a promising avenue for automated chatbot assessment, albeit with the acknowledgment that it is not yet a rigorous approach and requires further research to develop a comprehensive, standardized system (Chiang et al., 2023).

In the context of the automated generation system explored in this thesis, the Vicuna model is one of the two LLMs to be compared in performance for the task of rephrasing the potentially disfluent transcription aquired by Whisper and SHAS sentence by sentence, with the goal of getting a more fluent version. Its ability to produce detailed and well-structured responses holds promise in enhancing the quality of the generated content. The open-source nature of Vicuna, coupled with its promising performance, positions it as a interesting model to evaluate in the system, with potential towards helping to achieve the overarching goal of this thesis.

3.2.4. Llama 2

The LLaMA 2 model emerges as a potent and safe dialogue model that is optimized for dialogue applications, outperforming other models in terms of safety and helpfulness. Developed as an open-source initiative, it promotes collaboration and transparency, emphasizing a balanced approach to safety and helpfulness during the fine-tuning of models (Touvron et al., 2023).

The architecture of LLaMA 2 leverages pre-normalization in the transformer framework, introducing the Swigloo activation function and enhancing the context length. It explores various dimensions such as different context pre-training, depth of attention mechanisms, and the impact of feed-forward neural networks dimension on its performance. The training regimen suggests the potential of utilizing a cosine learning rate schedule to overcome limitations encountered during the training phase.

A notable feature of Llama 2 is the introduction of "ghost attention", a technique that synthetically integrates the system prompt at every level of the conversation, ensuring the initial instructions are consistently regarded. This, coupled with the use of concise instructions, aids in minimizing the token usage while maintaining focus on the prompt throughout the dialogue.

Despite its robustness in academic benchmarks, it exhibits a lower performance in solving math word problems compared to its counterpart, GPT-4. The fine-tuning process of the LLaMA model for chat applications involves a series of sophisticated techniques including alignment strategies, reward modeling, and a focus on data quality, emphasizing the efficacy of a smaller, clean dataset for SFT.

The LLaMA 2 model underscores a critical trade-off between safety and helpfulness, advocating for individual autonomy in determining the balance, as opposed to corporate dictation. It fosters a responsible approach to potentially hazardous information, rejecting requests for instructions on handling explosives, thereby indicating a responsible approach to information dissemination (Touvron et al., 2023).

In the broader context of this thesis, the LLaMA 2 model stands as a promising alternative to Vicuna in the automated generation system. The concrete model that will be used in the following for rephrasing is the "Llama-2-13b-chat-hf" version of the released models. Its open-source nature and emphasis on transparency make it a promising candidate for further exploration and integration into the system, aligning with the overarching goal of this thesis.

3.2.5. Bark

The Bark model, developed by Suno AI and available in the repository suno-ai/bark, represents a significant advancement in the field of TTS technologies. Unlike the other models like SpeechT5 (Ao et al., 2022), Bark has the capability to directly generate raw speech waveforms, thereby obviating the need for a separate vocoder during the inference stage. This is achieved through the innovative use of Encodec, a tool that functions both as a codec and a compression utility (Huggingface, 2023).

At its core, Bark is composed of four primary models that work in harmony to facilitate the generation of speech from text:

- **BarkSemanticModel (Text Model)**: A causal auto-regressive transformer model that processes tokenized text to predict semantic text tokens encapsulating the essence of the input text.
- **BarkCoarseModel (Coarse Acoustics Model)**: This model takes the output of the BarkSemanticModel and predicts the initial two audio codebooks necessary for Encodec.
- **BarkFineModel (Fine Acoustics Model)**: A non-causal autoencoder transformer that iteratively predicts the remaining codebooks based on the cumulative embeddings of the preceding codebooks.
- **EncodecModel**: Utilizes all the predicted codebook channels to decode the output audio array.

Each of these models can support conditional speaker embeddings, allowing the generated sound to be conditioned according to specific predefined voices.

Encodec stands central to Bark's efficiency, enabling the compression of audio into a lightweight format to diminish memory usage, and subsequently facilitating its decompression to retrieve the original audio. This process leverages eight codebooks, each comprising integer vectors that represent or embed the audio in integer form. Each successive codebook enhances the quality of audio reconstruction derived from the preceding ones. These codebooks can be learned efficiently by transformer models, a task Bark is specifically designed to accomplish.

Bark is highly versatile, capable of generating speech conditioned by a library of speaker embeddings accessible via its processor. It supports multilingual speech generation, including languages such as French and Chinese, without the necessity to specify the language explicitly. Moreover, Bark can produce non-verbal communications such as laughter, sighing, and crying, by incorporating corresponding cues in the input text. Remarkably, it can even generate music by enclosing words within musical notes (Huggingface, 2023).

Furthermore, Bark facilitates batch processing, allowing multiple text entries to be processed simultaneously, albeit at the cost of increased computational demand. This feature can potentially expedite the overall generation process on certain hardware configurations, such as GPUs.

In conclusion, the Bark model appears as a highly controllable and versatile tool in the TTS landscape, promising to improve speech generation with its innovative architecture and functionalities. Its role in the broader perspective of this thesis could be central to achieving high-fidelity and efficient speech generation, aligning perfectly with our goal.

3.3. Datasets

The datasets curated for this thesis play a great role in training and evaluating the automated generation system. These datasets are split into two categories: one representing the usual lecture style characterized by disfluencies and suboptimal sentence structures, and the other representing the target style which is characterized by well-articulated, fluent speech typically found in well-prepared TED talks.

Original Lecture Style Dataset

The dataset representing the original lecture style has been meticulously handcrafted, leveraging a rich array of resources to encompass a diverse set of linguistic patterns and styles. Primarily, the dataset incorporates lectures from the “Lecture Translator at KIT”, predominantly featuring recordings from the advanced AI lecture series, along with select recordings of lectures from different departments other than computer science, contained in the “KITOpen Catalogue”. This unfortunately leads to a substantial overrepresentation of computer science discourse in the dataset, due to the limited open availability of lecture recordings at KIT, which is why this will need to be addressed in the training of the style classifier later on.

Furthermore, the dataset is enriched with presentations from the “ISCA SIGSLT Seminars”, a series that brings forth research presentations and discussions in the field of computer science, often characterized by disfluencies and a considerable presence of non-native speakers, thereby adding a layer of complexity and diversity to the dataset.

Target Style Dataset

The target style dataset, on the other hand, aims to represent a more refined and fluent style of speech, akin to the one observed in well-prepared TED talks. To this end, the

MuST-C dataset has been employed, a resource that stands as a benchmark in the realm of multilingual speech translation corpora.

The MuST-C dataset, as detailed in a paper by Di Gangi et al. (2019), is a robust corpus that facilitates multilingual speech translation tasks. It encompasses a rich collection of TED talks, offering high-quality transcriptions aligned with their translations across multiple languages. The dataset is characterized by well-structured sentences, a coherent narrative style, and a minimal presence of disfluencies, making it an ideal representation of the target style for this thesis.

The utilization of the MuST-C dataset promises to lead the style classifier of the automated generation system to recognize sentences that are not only fluent but also resonate with the eloquence and structured narrative style that is synonymous with TED talks.

Conclusion

In conclusion, the curated datasets stand as a backbone in the development and fine-tuning of the style classifier for evaluating the automated generation system explored in this thesis. The selection and curation process ensures a diverse dataset, facilitating a comprehensive training regimen for the models involved. Due to limited availability of lecture recordings from different departments at KIT, the distribution of topics in the lecture style dataset is skewed towards computer science content, which will be later be addressed. Additionally, the MuST-C dataset is trimmed by a considerable amount to ensure an approximate 50:50 split with the entire dataset containing 40645 sentences.

3.4. Evaluation Metrics

In the next few subsections, several metrics being used to evaluate the sentence transformation in this research will be introduced and the way they function will be explained.

3.4.1. Style Classification with DeBERTa

In the process of style classification, we leverage the capabilities of decoding-enhanced BERT with disentangled attention (DeBERTa) v3, a state-of-the-art feature extractor renowned for its performance in numerous natural language understanding tasks. The architecture of DeBERTa v3, as detailed by He et al. (2023), operates based on a deep learning framework that enhances the understanding of the inter-relationships between different words in a sentence, thereby enabling a more nuanced analysis of textual data.

Initially, we train a baseline variant of DeBERTa by jointly training it with a linear classification layer. However, this approach primarily capitalizes on the content skewness of the data for classification, rather than focusing on the style, a limitation that becomes evident in the result section 4.1.

To address this, we adopt two strategies aimed at training the style classifier and DeBERTa as a feature extractor, such that the features derived are more aligned with style rather than content. This involves the incorporation of a content classification layer that identifies the recording source of a sentence in the dataset based on the features from DeBERTa. This layer undergoes training alternately with the style classification layer and DeBERTa combined, introducing an additional loss term to the style loss to encourage the reduction of performance of the content classification layer through DeBERTa layer adjustments.

In the first approach, we employ an adversarial loss coupled with a gradient reversal layer (Ganin et al., 2016). This strategy directs the training to reduce the emphasis DeBERTa places on features beneficial for content classification, thereby fostering a more style-centric focus. The concrete approach draws inspiration from the one described in the appendix of a paper by Liu and Niehues (2022).

The second strategy involves the addition of Kullback-Leibler (KL) Divergence between the content classifier distribution and a uniform distribution across all classes. The KL Divergence, a measure of how one probability distribution diverges from a second, expected probability distribution, aids in promoting the same objective as the first approach, steering the focus towards style over content.

Through these refined approaches, we aspire to develop a style classifier that is not only adept at distinguishing styles but also mitigates the influence of content skewness, paving the way for a more balanced and effective style classification system.

3.4.2. BLEURT

In the realm of text generation, the evaluation of generated content is an important aspect. The BLEURT metric, which stands for “BERT Language Understanding for Evaluating with Reinforced Tuning”, emerges as a robust tool in this context, offering a learned metric that leverages the power of BERT to model human judgments on text generation tasks (Sellam et al., 2020).

Developed by Google Research, BLEURT aims to overcome the limitations of existing metrics such as BLEU and ROUGE, which often do not correlate well with human judgments. The metric is designed to be both expressive, utilizing available ratings data for training, and robust to distribution drifts, ensuring effective extrapolation in diverse scenarios.

A distinctive feature of BLEURT is its pre-training scheme, which leverages large volumes of synthetic data created through various techniques including mask-filling with BERT, backtranslation, and random word dropping. This pre-training encompasses several tasks that capture a wide array of lexical and semantic differences, utilizing signals such as BLEU, ROUGE, BERTscore, backtranslation likelihood, and entailment signals, among others. This approach aids in the generalization of BLEURT, enabling it to offer state-of-the-art results on recent WMT Metrics shared tasks and the WebNLG competition dataset, especially when the training data is scarce and out-of-distribution (Sellam et al., 2020).

The BLEURT metric has demonstrated remarkable performance and robustness across different tasks and datasets, showcasing its potential as a valuable tool in the field of natural language generation. In the broader perspective of this thesis, incorporating BLEURT in the evaluation metrics could offer a nuanced understanding of the quality of generated content, aiding in the evaluation of the models involved to achieve the desired level of fluency and content preservation in the lecture recordings.

3.4.3. COMET

The COMET framework, initially developed for reference-based MT evaluation, played a central role in the joint contribution of Instituto Superior Técnico (IST) and Unbabel to the WMT 2022 Quality Estimation (QE) shared task (Rei et al., 2022).

This collaboration led to the creation of COMETKIWI, a system that integrates the strengths of COMET and OPENKIWI, the latter being known for its proficiency in word and sentence-level QE (Kepler et al., 2019). The system was designed to be adept in multilingual generalization, capable of handling unseen languages through few-shot training, and showcased a remarkable performance in the WMT 2022 QE shared task, notably achieving a 0.572 Spearman correlation in sentence-level Direct Assessments (DAs) (Zerva et al., 2022).

A significant innovation introduced was a new interpretability method that combines attention and gradient information, further refined through a head-level scalar mix module, enhancing the system's ability to delineate the relevance of attention heads effectively.

In the context of this thesis, the reference-free variant of the COMET metric could be a pivotal tool in evaluating the quality of generated content, aligning well with the research's primary objective of enhancing the fluency and content preservation in automated lecture recordings.

3.4.4. SentenceTransformers

SentenceTransformers is a Python framework developed to facilitate the training and utilization of sentence, paragraph, and image embeddings that are compatible with BERT-based models (Reimers and Gurevych, 2019). The framework is designed to work efficiently with large texts, offering a significant speed-up compared to the traditional BERT models when it comes to embedding sentences.

A notable feature of SentenceTransformers is its support for various training approaches, including supervised, unsupervised, and multi-task learning. This flexibility allows researchers and practitioners to tailor the training process to suit specific requirements, thereby optimizing the performance of the embedding models for different tasks.

In the context of this research, SentenceTransformers can be employed to generate embeddings for the sentences in the lecture recordings. These embeddings can then be used to analyze the semantic similarity between sentences, facilitating a more nuanced understanding of the content and aiding in the evaluation of the quality of the generated content.

By leveraging the capabilities of SentenceTransformers, it is possible to develop more sophisticated evaluation metrics that can effectively gauge the content preservation in automated lecture recordings. The framework offers a robust toolset for working with sentence embeddings, providing a foundation for exploring innovative approaches to evaluating natural language generation tasks (Reimers and Gurevych, 2019).

In summary, SentenceTransformers stands as a significant tool in the evaluation metrics section, offering a pathway to develop metrics that are both robust and finely tuned to the specific requirements of the research. Its integration into the evaluation process promises to enhance the accuracy and reliability of the assessments, contributing to the achievement of the research objectives. The concrete model that was chosen for the experiments is the "all-mpnet-base-v2" due to it having the highest performance and as a similarity measurement, cosine-similarity was used.

4. Results

In this chapter, we delve into the analysis of the results derived from the experiments laid out in the preceding chapter. The results are provide insights into understanding the effectiveness of the various approaches and methodologies employed in this research. They offer a lens through which we can scrutinize the performance of different techniques in enhancing the fluency and structure of academic lecture recordings.

Throughout the discussion of the results, we will frequently refer to two distinct styles: "style 0" and "style 1". To provide a clear understanding, it is of great importance to define these terms at the outset. When we refer to "style 0", we are pointing to the original, potentially disfluent, "lecture style". This style embodies the raw, unaltered transcripts derived from academic lectures, retaining the original nuances, including possible disfluencies. On the other hand, "style 1" denotes the target "TED Talk style". This style is characterized by a more polished, fluent, and structured manner of presentation, akin to the speeches delivered in TED Talks. The transformation from "style 0" to "style 1" is at the heart of our research, aiming to elevate the quality of lecture transcripts to a level that mirrors the eloquence and fluency found in TED Talks.

As we navigate through the results, we will analyze the performance metrics of different style classifications and transformation techniques, shedding light on their efficacy in achieving the desired style transfer while preserving the core content. The insights garnered from this analysis will be instrumental in answering the research questions posited in the introductory chapter, thereby fulfilling the central objective of this thesis.

4.1. Style Classification

In this section, we focus on the training of style-specific classifiers using adversarial training and KL divergence approaches. The training process, albeit fragile, has shown potential when appropriate hyperparameters are carefully selected, as detailed in appendices A.1 and A.2.

A critical observation during the training was the DeBERTa model's tendency to converge to a state of "forgetfulness" of its pre-training when subjected to high learning rates or excessive epochs. This resulted in a decline in the meaningfulness of the embedding outputs, with the style classifier's accuracy performance on the validation set approaching 0.5. Despite this, with proper fine-tuning, the DeBERTa model exhibited a heightened focus on style features compared to the baseline, albeit without significant improvements across all content categories. This suggests a noticeable content component in the embeddings that remains unaltered by the training approaches employed.

We further analyzed the style classification accuracy, focusing on various lecture categories including Machine Learning (ML), Database Systems (DBS), Biology (Bio), and History (Hist). The results, presented in Table 4.1, indicate an increase in accuracy for lectures outside the computer science domain, aligning with the objectives of our training concepts. However, it is essential to scrutinize the performance on the TED Talks target dataset labeled with ground truth "style 1" to ensure the gains are not merely due to a higher classification of sentences as "style 0".

Table 4.2 portrays the mean probabilities and accuracies derived from the target style dataset. Despite a slight decline in performance, especially with the adversarial approach, the data suggests a fulfillment of the training objective, with DeBERTa paying increased attention to style.

However, the sentence-level approach of our classification presents a possible challenge, since short sentences do might not portray any meaningful stylistic attributes, hinting at the potential benefits of incorporating longer text sequences for improved classification.

Method	ML	DBS	Bio	Hist
Baseline	72.27%	77.10%	33.29%	11.50%
Adversarial	78.66%	79.59%	45.65%	20.86%
KL Div	75.00%	78.20%	34.97%	14.66%

Table 4.1.: Style Classification Accuracy across two Samples each from different Lecture Categories

Method	Baseline	Adversarial	KL Div
Mean Probability for "Style 1"	96.68%	89.23%	94.72%
Accuracy (Percentage of Samples with "Style 1" Probability ≥ 0.5)	96.91%	91.09%	95.79%

Table 4.2.: Metrics of Style Classifiers on Target Style Dataset

4.2. Style Transfer Strength of Transformations

In this segment, we turn our attention to the evaluation of style strength performance, utilizing the style classifiers trained in the previous section.

By applying softmax to the logits of style predictions, we derive probabilities for each style class. This allows us to compare the mean probabilities between the original lecture dataset and each transformation across different models. Additionally, we quantify the strength of the style transfer by examining the percentage of samples that would be classified as having "style 1".

From the data presented, it becomes evident that the transformations are indeed effectuating a style transfer. However, the strength of this transfer appears to be somewhat subdued compared to initial expectations. This subdued performance could be attributed to several factors: limitations in the classifier, constraints arising from sentence-level operations, or the possibility that the style transfer capability was not as potent as initially hypothesized.

A comparative analysis between the different methods reveals that the LLaMA 2 model generally exhibits a more robust style transfer strength than Vicuna. Moreover, there are discernible differences in the efficacy of prompting methods across models. LLaMA 2, for instance, benefits considerably from the rephrased context, while Vicuna demonstrates optimal style transfer strength on our trained classifiers with plain zero-shot prompting.

Tables 4.3, 4.4, 4.5, 4.6, and 4.7 provide a comprehensive view of the results, offering insights into the performance metrics of different style classifications and transformation techniques.

Method	Baseline	Adversarial	KL Div
Mean Probability for "Style 1"	3.21%	8.24%	5.51%
Accuracy (Percentage of Samples with "Style 1" Probability ≥ 0.5)	3.17%	6.20%	4.63%

Table 4.3.: Metrics of Style Classifiers on Original Lecture Style Dataset

	Method			
Classifier	zero-shot	original context	rephrased context	few-shot
Baseline	14.29%	14.63%	15.81%	10.49%
Adversarial	19.63%	19.45%	20.34%	17.86%
KL Div	18.24%	18.19%	19.13%	15.26%

Table 4.4.: Mean Probabilities for "Style 1" of Transformations on Original Lecture Style Dataset (LLaMA 2)

Classifier	Method			
	zero-shot	original context	rephrased context	few-shot
Baseline	14.06%	14.56%	15.77%	10.47%
Adversarial	17.31%	17.35%	18.11%	15.48%
KL Div	17.11%	17.25%	18.17%	14.30%

Table 4.5.: Percentage of Samples with "Style 1" Probability ≥ 0.5 of Transformations on Original Lecture Style Dataset (LLaMA 2)

Classifier	Method			
	zero-shot	original context	rephrased context	few-shot
Baseline	12.40%	12.34%	12.13%	9.76%
Adversarial	18.09%	17.34%	17.29%	16.63%
KL Div	16.30%	15.33%	15.49%	14.29%

Table 4.6.: Mean Probabilities for "Style 1" of Transformations on Original Lecture Style Dataset (Vicuna)

Classifier	Method			
	zero-shot	original context	rephrased context	few-shot
Baseline	12.31%	12.24%	12.00%	9.65%
Adversarial	15.90%	15.09%	14.98%	14.37%
KL Div	15.42%	14.51%	14.65%	13.38%

Table 4.7.: Percentage of Samples with "Style 1" Probability ≥ 0.5 of Transformations on Original Lecture Style Dataset (Vicuna)

4.3. Content Preservation of Transformations

In the final section of this chapter, we focus on the essential aspect of content preservation during the style transfer process. To gauge the efficacy of content preservation, we employ three distinct methods, as explained in section 3.4. These methods generate scores predominantly ranging between 0 and 1 (with possible exceptions of scores close to, but outside of, the interval with BLEURT and COMET), where a score of 1 signifies optimal performance, and a score of 0 indicates poor performance.

Initially, we examine the results obtained when the original sentences are used both as predictions and references, aiming to observe a high mean score over the original lecture dataset, since the content preservation score should be highest when using the same sentence as the predictions. This approach reveals the inherent strengths and weaknesses of different scoring metrics, with BLEURT and SentenceTransformer demonstrating high scores, indicative of being a good content preservation metric for our means. In contrast, the COMET model, optimized for quality estimation of machine translation systems, exhibits a lower score, suggesting it may not be the ideal choice for our specific use case. Interestingly, a brief investigation into the COMET scores revealed a considerable variability when the same sentence was used for both prediction and reference. For instance, the sentence "Hello, my name is Thomas." receives a score of 0.8625, while the phrase "And he will also give part of the lecture." is assigned a substantially lower score of 0.3486.

As we delve deeper, the data unveils that content preservation is generally well-maintained across all transformations. A notable observation is the slightly superior performance of Vicuna over Llama 2 in safeguarding the content. When we turn our lens to the prompting methods, it emerges that the few-shot method stands out in preserving content, a result of utilizing handcrafted sample sentences that retain the original meaning while enhancing sentence structure and fluency. Among the zero-shot methods, the absence of context appears to foster the best preservation of meaning.

Tables 4.8, 4.9, and 4.10 encapsulate the detailed results, offering a granular view of the performance of different methods in preserving content during the style transfer process.

Score	predictions = references
BLEURT	0.9733
COMET	0.2843
SentenceTransformer	1.0

Table 4.8.: Content Preservation Scores of Original Lecture Style Dataset (using Orginal Sentence as both prediction and reference)

	Method			
Score	zero-shot	original context	rephrased context	few-shot
BLEURT	0.7422	0.7190	0.7094	0.7693
COMET	0.3087	0.3090	0.3106	0.2928
SentenceTransformer	0.8199	0.7856	0.7728	0.8666

Table 4.9.: Content Preservation Scores of Transformations on Original Lecture Style Dataset (LLaMA 2)

	Method			
Score	zero-shot	original context	rephrased context	few-shot
BLEURT	0.7520	0.7360	0.7364	0.7729
COMET	0.3054	0.3052	0.3057	0.2934
SentenceTransformer	0.8372	0.8130	0.8133	0.8668

Table 4.10.: Content Preservation Scores of Transformations on Original Lecture Style Dataset (Vicuna)

5. Conclusion

To conclude this research, this chapter will give concise answers to the research questions posed in the introductory chapter 1. Furthermore, potential avenues for future research are being pointed out.

5.1. Answers to Research Questions

Here, we revisit the research questions from section 1.3.

Research Question 1: *How can different prompting techniques influence the effectiveness of LLMs in enhancing the fluency and sentence structure of academic lecture recordings, with regards to style transfer strength and content preservation?*

The exploration undertaken in this research clearly affirms the potential of LLMs in transforming sentences from potentially disfluent lecture recordings into more fluent and structurally sound versions. Our analysis indicated a consistent increase in the mean probability for the target style across all transformations, showcasing the efficacy of the applied techniques in enhancing style transfer strength.

In a detailed examination of individual models, LLaMA 2 exhibited optimal performance with zero-shot prompting when supplemented with the context of three preceding rephrased sentences. Conversely, Vicuna demonstrated superior results utilizing a zero-shot prompt without context. This divergence in optimal conditions for each model underscores the necessity for tailored approaches in leveraging different LLMs.

When evaluating content preservation, the few-shot method emerged as the most effective, particularly when utilizing handcrafted rephrased sentences as examples in our experiment. This strategy, which focused on minimal alterations to maintain the core content while enhancing fluency and structure, proved to be highly effective.

Overall, while LLaMA 2 generally outperformed in terms of style transfer strength, Vicuna maintained a higher average in content preservation, indicating a balanced proficiency in both critical aspects of the transformation process.

Research Question 2: *How can a style classifier be trained to prioritize style over content in extracting features for dense embeddings, especially in the presence of skewed content distribution in the training data?*

The training of a style classifier to prioritize style over content presented a delicate and intricate process, necessitating meticulous hyperparameter tuning. Our approach leveraged adversarial training and KL divergence to foster a focus on style-centric features, albeit with a recognition of the inherent fragility in the training process.

A critical observation was the propensity for the DeBERTa model to converge to a less informative state with excessive training epochs or high learning rates, a state characterized by a loss of pre-training insights and diminished embedding output significance. This phenomenon manifested in a convergence of the style classifier's accuracy performance on the validation set towards a 0.5 benchmark, indicating a neutral performance.

Despite these challenges, with judicious training, the DeBERTa model exhibited a heightened focus on style features compared to the baseline, albeit with room for further optimization to reduce the residual content component in the embeddings.

5.2. Future Work

While many introductory questions on how to realize a system that takes potentially disfluent lecture recordings and generates a fluent version from that have been addressed in this thesis, there is still a great potential for further exploring interesting research directions.

To begin with, all the transformations of the transcribed lecture content have been performed on the sentence level in this thesis. Although part of our experiments already incorporated some context in the form of the preceding sentences, one possible avenue to investigate would be to use longer sections of text, or potentially entire paragraphs in one single transformation, allowing the LLM to have more possibilities for restructuring the content for enhanced clarity and fluency. One issue that probably would need to be addressed there is the need for generating longer token sequences than possible by default with the models being used in this thesis with the Hugging Face Inference API, and also, depending on the length of the text sections to be transformed, potential problems with the context window that can be taken into account when generating the sequence.

Secondly, one could further investigate how to make the LLMs better follow the given instructions. In our experiments, a lot of prompt engineering was necessary in order for the LLMs to adhere to the instructions given in the prompt. For example, likely due to being trained by RLHF to be a helpful assistant, the variant of LLaMA 2 we used often gave an answer to questions that were part of the text to be transformed in the generated text, instead of simply rephrasing them while preserving the meaning, as it was instructed to do. One way this could be addressed is to use better performing models with a higher parameter count, since they tend to also perform better with following instructions, although this approach would require higher amounts of GPU memory. If such resources are not available, another concept to explore is using classifier-free guidance for the text generation in LLMs, a concept inspired by the technique of the same name in text-to-image generation. This approach can successfully help in "increas[ing] the faithfulness and coherence of assistants” and “brings improvements equivalent to a model with twice the parameter-count”, as shown by Sanchez et al. (2023).

At last, future work remains to be explored not only in improving the capabilities of the LLMs to provide accurate and content-preserving style transfer, but also in evaluating this transformations more precisely. A promising idea, although requiring greater amounts of manual labour, would be to assess the transformed transcripts by taking human evaluations into consideration, since despite the fact that automatic evaluation methods keep improving, no metric can consistently come close to human judgement yet, especially when it comes to reviewing the style of natural language. When having acquired data in the form of human evaluation, one could also work on creating a new automatic evaluation metric that has high correlation with human judgements.

Bibliography

Ao, Junyi, Rui Wang, Long Zhou, Chengyi Wang, Shuo Ren, Yu Wu, Shujie Liu, Tom Ko, Qing Li, Yu Zhang, Zhihua Wei, Yao Qian, Jinyu Li, and Furu Wei (2022). SpeechT5: Unified-Modal Encoder-Decoder Pre-Training for Spoken Language Processing.

Baevski, Alexei, Henry Zhou, Abdelrahman Mohamed, and Michael Auli (2020). Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems.* NIPS'20. Vancouver, BC, Canada: Curran Associates Inc. ISBN: 9781713829546.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, ed. by Yoshua Bengio et al.

Bai, Yuntao, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan (2022). Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback.

Borsos, Zalán, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour (2023). AudioLM: a Language Modeling Approach to Audio Generation. arXiv: `2209.03143 [cs.SD]`.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pages 1877–1901.

Chan, William, Navdeep Jaitly, Quoc Le, and Oriol Vinyals (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964.

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). Long Short-Term Memory-Networks for Machine Reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pages 551–561.

Chiang, Wei-Lin, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing (2023). Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality.

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pages 1724–1734.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pages 4171–4186.

Di Gangi, Mattia A., Roldano Cattoni, Luisa Bentivogli, Matteo Negri, and Marco Turchi (2019). MuST-C: a Multilingual Speech Translation Corpus. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pages 2012–2017.

Fu, Zhenxin, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan (2018). Style Transfer in Text: Exploration and Evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1.

Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky (2016). Domain-Adversarial Training of Neural Networks.

He, Pengcheng, Jianfeng Gao, and Weizhu Chen (2023). DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing.

Hinterleitner, Florian (2017). Quality of Synthetic Speech. ISBN: 978-981-10-3733-7.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). Denoising Diffusion Probabilistic Models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc. ISBN: 9781713829546.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). Multilayer feedforward networks are universal approximators. In *Neural Networks* 2.5, pages 359–366. ISSN: 0893-6080.

Howard, Jeremy and Sebastian Ruder (2018). Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers). Melbourne, Australia: Association for Computational Linguistics, pages 328–339.

Huggingface (2023). Pre-trained models for text-to-speech. Accessed: 2023-09-11. URL: `https://huggingface.co/learn/audio-course/chapter6/pre-trained_models`.

Jin, Di, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea (2022). Deep Learning for Text Style Transfer: A Survey. In *Computational Linguistics* 48.1, pages 155–205.

Juang, B. and Lawrence Rabiner (2005). Automatic Speech Recognition - A Brief History of the Technology Development.

Jurafsky, Dan and James H. Martin (2023). Speech and Language Processing (3rd ed. draft).

Kepler, Fabio, Jonay Trénous, Marcos Treviso, Miguel Vera, and André F. T. Martins (2019). OpenKiwi: An Open Source Framework for Quality Estimation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations.* Florence, Italy: Association for Computational Linguistics, pages 117–122.

Kiefer, J. and J. Wolfowitz (1952). Stochastic Estimation of the Maximum of a Regression Function. In *The Annals of Mathematical Statistics* 23.3, pages 462–466.

Klakow, Dietrich and Jochen Peters (2002). Testing the Correlation of Word Error Rate and Perplexity. In *Speech Commun.* 38.1, pages 19–28. ISSN: 0167-6393.

Krishna, Kalpesh, John Wieting, and Mohit Iyyer (2020). Reformulating Unsupervised Style Transfer as Paraphrase Generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Online: Association for Computational Linguistics, pages 737–762.

Kudo, Taku (2018). Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Melbourne, Australia: Association for Computational Linguistics, pages 66–75.

Kudo, Taku and John Richardson (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* Brussels, Belgium: Association for Computational Linguistics, pages 66–71.

Lakew, Surafel Melaku, Mauro Cettolo, and Marcello Federico (2018). A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation. In *Proceedings of the 27th International Conference on Computational Linguistics.* Santa Fe, New Mexico, USA: Association for Computational Linguistics, pages 641–652.

Li, Juncen, Robin Jia, He He, and Percy Liang (2018). Delete, Retrieve, Generate: a Simple Approach to Sentiment and Style Transfer. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers).* New Orleans, Louisiana: Association for Computational Linguistics, pages 1865–1874.

Liu, Danni and Jan Niehues (2022). Learning an Artificial Language for Knowledge-Sharing in Multilingual Translation. In *Proceedings of the Seventh Conference on Machine Transla-*

tion (WMT). Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, pages 188–202.

McCulloch, Warren S. and Walter Pitts (1943). A logical calculus of the ideas immanent in nervous activity. In *The bulletin of mathematical biophysics* 5.4, pages 115–133. ISSN: 1522-9602.

Mir, Remi, Bjarke Felbo, Nick Obradovich, and Iyad Rahwan (2019). Evaluating Style Transfer for Text. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers).* Minneapolis, Minnesota: Association for Computational Linguistics, pages 495–504.

Oord, Aäron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). WaveNet: A Generative Model for Raw Audio. In *Arxiv*.

Radford, Alec, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever (2023). Robust Speech Recognition via Large-Scale Weak Supervision. In *Proceedings of the 40th International Conference on Machine Learning*, ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, pages 28492–28518.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). Improving Language Understanding by Generative Pre-Training.

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. In *J. Mach. Learn. Res.* 21.1. ISSN: 1532-4435.

Rei, Ricardo, Marcos Treviso, Nuno M. Guerreiro, Chrysoula Zerva, Ana C Farinha, Christine Maroti, José G. C. de Souza, Taisiya Glushkova, Duarte Alves, Luisa Coheur, Alon Lavie, and André F. T. Martins (2022). CometKiwi: IST-Unbabel 2022 Submission for the Quality Estimation Shared Task. In *Proceedings of the Seventh Conference on Machine Translation (WMT).* Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, pages 634–645.

Reimers, Nils and Iryna Gurevych (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).* Hong Kong, China: Association for Computational Linguistics, pages 3982–3992.

Roux, Thibault Bañeras, Mickael Rouvier, Jane Wottawa, and Richard Dufour (2022). Qualitative Evaluation of Language Model Rescoring in Automatic Speech Recognition. In *Proc. Interspeech 2022*, pages 3968–3972.

Rugayan, Janine, Torbjørn Svendsen, and Giampiero Salvi (2022). Semantically Meaningful Metrics for Norwegian ASR Systems. In *Proc. Interspeech 2022*, pages 2283–2287.

Sanchez, Guillaume, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman (2023). Stay on topic with Classifier-Free Guidance.

Sellam, Thibault, Dipanjan Das, and Ankur Parikh (2020). BLEURT: Learning Robust Metrics for Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* Online: Association for Computational Linguistics, pages 7881–7892.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Berlin, Germany: Association for Computational Linguistics, pages 1715–1725.

Sindel, Aline, Abner Hernandez, Seung Hee Yang, Vincent Christlein, and Andreas Maier (2022). SliTraNet: Automatic Detection of Slide Transitions in Lecture Videos using Convolutional Neural Networks.

Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models.

Tsiamas, Ioannis, Gerard I. Gállego, José A. R. Fonollosa, and Marta R. Costa-jussà (2022). SHAS: Approaching optimal Segmentation for End-to-End Speech Translation.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems,* ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc.

Viswanathan, Mahesh and Madhubalan Viswanathan (2005). Measuring speech quality for text-to-speech systems: development and assessment of a modified mean opinion score (MOS) scale. In *Computer Speech & Language* 19.1, pages 55–83. ISSN: 0885-2308.

Yu, Dong and Li Deng (2014). Automatic Speech Recognition: A Deep Learning Approach. Springer Publishing Company, Incorporated. ISBN: 1447157788.

Zen, Heiga, Keiichi Tokuda, and Alan W. Black (2009). Statistical parametric speech synthesis. In *Speech Communication* 51.11, pages 1039–1064. ISSN: 0167-6393.

Zerva, Chrysoula, Frédéric Blain, Ricardo Rei, Piyawat Lertvittayakumjorn, José G. C. de Souza, Steffen Eger, Diptesh Kanojia, Duarte Alves, Constantin Orăsan, Marina Fomicheva, André F. T. Martins, and Lucia Specia (2022). Findings of the WMT 2022 Shared Task on Quality Estimation. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, pages 69–99.

A. Appendix

A.1. Adversarial Training Code

```python
import os
from itertools import chain

os.environ["CUDA_VISIBLE_DEVICES"] = "1"

import sklearn
import evaluate
import torch
from datasets import load_dataset
from tensorboardX import SummaryWriter
from torch import nn
from torch.autograd import Function
from torch.optim import AdamW
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts
from torch.utils.data import DataLoader
from tqdm.auto import tqdm
from transformers import AutoModel, AutoTokenizer, DataCollatorWithPadding

data_files = {"train": "train.csv",
              "validation": "dev.csv", "test": "test.csv"}
raw_datasets = load_dataset("csv", data_files=data_files)
checkpoint = "microsoft/deberta-v3-base"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)


def tokenize_function(example):
    return tokenizer(example["Sentence"], truncation=True)


tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

tokenized_datasets = tokenized_datasets.remove_columns(
    ["Sentence Index", "Sentence", "Split"])
tokenized_datasets = tokenized_datasets.rename_column(
    "Style Label", "style_labels")
tokenized_datasets = tokenized_datasets.rename_column(
    "Content Label", "content_labels")
tokenized_datasets.set_format("torch")

train_dataloader = DataLoader(
    tokenized_datasets["train"], shuffle=True, batch_size=16, collate_fn=data_collator
)
eval_dataloader = DataLoader(
    tokenized_datasets["validation"], batch_size=32, collate_fn=data_collator
)


class GradientReversal(Function):
    @staticmethod
    def forward(ctx, x, alpha):
        ctx.save_for_backward(x, alpha)
        return x

    @staticmethod
    def backward(ctx, grad_output):
        grad_input = None
        _, alpha = ctx.saved_tensors
        if ctx.needs_input_grad[0]:
            grad_input = - alpha*grad_output
        return grad_input, None


revgrad = GradientReversal.apply


class GradientReversal(nn.Module):
    def __init__(self, alpha):
        super().__init__()
        self.alpha = torch.tensor(alpha, requires_grad=False)

    def forward(self, x):
        return revgrad(x, self.alpha)


class AdversarialModel(nn.Module):
    def __init__(self, alpha=1.0):
        super().__init__()
        self.deberta = AutoModel.from_pretrained(checkpoint)
        self.style_classifier = nn.Linear(self.deberta.config.hidden_size, 2)
        self.dropout = nn.Dropout(self.deberta.config.hidden_dropout_prob)
        self.gradient_reversal = GradientReversal(alpha)
        self.content_classifier = nn.Linear(
            self.deberta.config.hidden_size, 208)

        # Freeze all parameters
        for param in self.deberta.parameters():
            param.requires_grad = False

        # Unfreeze the last layers
        for i in range(-1, -7, -1):
            for param in self.deberta.encoder.layer[i].parameters():
                param.requires_grad = True

    def forward(self, input_ids, attention_mask):
        outputs = self.deberta(input_ids, attention_mask=attention_mask)
        pooled_output = outputs.last_hidden_state[:, 0, :]
        pooled_output = self.dropout(pooled_output)
        style_logits = self.style_classifier(pooled_output)
        reversed_output = self.gradient_reversal(pooled_output)
        content_logits = self.content_classifier(reversed_output)
        return style_logits, content_logits


model = AdversarialModel()

base_lr = 5e-5
decay_factor = 0.75

# Get the layers of the DeBERTa model
layers_to_train = model.deberta.encoder.layer[-1:-7:-1]

# Calculate learning rates for each layer
layerwise_lr = [base_lr * (decay_factor ** i) for i in range(len(layers_to_train))]

# Create parameter groups with the calculated learning rates
param_groups = []
for lr, layer in zip(layerwise_lr, layers_to_train):
    param_groups.append({'params': layer.parameters(), 'lr': lr})

# Add other model parameters (e.g., style_classifier) with the base learning rate
param_groups.append({'params': model.style_classifier.parameters(), 'lr': base_lr*10})

content_optimizer = AdamW(model.content_classifier.parameters(), lr=base_lr*10, eps=1e-6)
style_optimizer = AdamW(param_groups)

T_0 = 6 # Number of epochs before the first restart
content_scheduler = CosineAnnealingWarmRestarts(content_optimizer, T_0=T_0)
style_scheduler = CosineAnnealingWarmRestarts(style_optimizer, T_0=T_0)


# Retrieve the current learning rates of the optimizer.
def get_current_lr(optimizer):
    lrs = [pg['lr'] for pg in optimizer.param_groups]
    return lrs


num_epochs = 120
num_training_steps = num_epochs * len(train_dataloader)

device = torch.device(
    "cuda") if torch.cuda.is_available() else torch.device("cpu")
print(device)
model.to(device)

progress_bar = tqdm(range(num_training_steps))

run = 'adversarial_training_1.19'

```

```
writer = SummaryWriter(log_dir=f'runs/{run}')

metrics = {
    "accuracy": evaluate.load("accuracy"),
    "f1": evaluate.load("f1"),
    "precision": evaluate.load("precision"),
    "recall": evaluate.load("recall"),
}

model.train()


def evaluate_model(model, dataloader, metrics):
    model.eval()
    total_style_loss = 0
    total_content_loss = 0
    style_preds = []
    style_labels = []
    content_preds = []
    content_labels = []

    with torch.no_grad():
        for batch in dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            style_logits, content_logits = model(
                batch["input_ids"], batch["attention_mask"])

            style_loss = nn.CrossEntropyLoss()(
                style_logits, batch["style_labels"])
            content_loss = nn.CrossEntropyLoss()(
                content_logits, batch["content_labels"])
            total_style_loss += style_loss.item()
            total_content_loss += content_loss.item()
            style_preds.extend(style_logits.argmax(dim=1).cpu().numpy())
            content_preds.extend(content_logits.argmax(dim=1).cpu().numpy())
            style_labels.extend(batch["style_labels"].cpu().numpy())
            content_labels.extend(batch["content_labels"].cpu().numpy())

    results = {}
    avg_style_loss = total_style_loss / len(dataloader)
    avg_content_loss = total_content_loss / len(dataloader)
    results['Loss/style'] = avg_style_loss
    results['Loss/content'] = avg_content_loss
    for name, metric in metrics.items():
        if (name == 'accuracy'):
            style_result = metric.compute(
                predictions=style_preds, references=style_labels)
            content_result = metric.compute(
                predictions=content_preds, references=content_labels)
        else:
            style_result = metric.compute(
                predictions=style_preds, references=style_labels, average='micro')
            content_result = metric.compute(
                predictions=content_preds, references=content_labels, average='micro')
```

```python
        results[f'{name}/style'] = style_result[name]
        results[f'{name}/content'] = content_result[name]

    model.train()
    return results


gradient_accumulation_steps = 4

avg_style_loss = 0
avg_content_loss = 0
avg_combined_loss = 0

for epoch_block in range(0, num_epochs, 12): # Alternating training of objectives
    for objective in range(2):  # Two objectives: style and content
        for sub_epoch in range(6):  # Train each objective
            epoch = epoch_block + objective*6 + sub_epoch
            results = evaluate_model(model, eval_dataloader, metrics)
            print(f'Evaluation Results Epoch {epoch}:', results, flush=True)  # Print the
    results
            writer.add_scalar('Loss/style_val', results['Loss/style'], epoch)
            writer.add_scalar('Loss/content_val', results['Loss/content'], epoch)

            # Log metrics to TensorBoard
            for name, value in results.items():
                writer.add_scalar(f'{name}_val', value, epoch)

            total_style_loss = 0
            total_content_loss = 0
            total_combined_loss = 0

            if objective == 0:
                # Content classifier update
                for step, batch in enumerate(train_dataloader):
                    batch = {k: v.to(device) for k, v in batch.items()}
                    input_ids = batch["input_ids"]
                    attention_mask = batch["attention_mask"]
                    style_logits, content_logits = model(
                        input_ids=input_ids, attention_mask=attention_mask)
                    content_loss = nn.CrossEntropyLoss()(
                        content_logits, batch["content_labels"])
                    style_loss = nn.CrossEntropyLoss()(
                        style_logits, batch["style_labels"])
                    total_content_loss += content_loss.item()
                    total_style_loss += style_loss.item()

                    # Adversarial loss for content
                    content_probs = torch.softmax(content_logits, dim=-1)
                    content_target_probs = content_probs[range(
                        content_logits.shape[0]), batch["content_labels"]]
                    adv_content_loss = torch.log(1 - content_target_probs).mean()
                    combined_loss = style_loss + 0.1 * adv_content_loss
                    total_combined_loss += combined_loss.item()

                    content_loss = content_loss / gradient_accumulation_steps
                    content_loss.backward()

                    if (step + 1) % gradient_accumulation_steps == 0:
                        content_optimizer.step()
                        content_optimizer.zero_grad()
                        content_scheduler.step(epoch + step / len(train_dataloader))
                    progress_bar.update(1)

            else:
                # Style classifier and DeBERTa update
                for step, batch in enumerate(train_dataloader):
                    batch = {k: v.to(device) for k, v in batch.items()}
                    input_ids = batch["input_ids"]
                    attention_mask = batch["attention_mask"]
                    style_logits, content_logits = model(
                        input_ids=input_ids, attention_mask=attention_mask)
                    style_loss = nn.CrossEntropyLoss()(
                        style_logits, batch["style_labels"])
                    content_loss = nn.CrossEntropyLoss()(
                        content_logits, batch["content_labels"])
                    total_style_loss += style_loss.item()
                    total_content_loss += content_loss.item()

                    # Adversarial loss for content
                    content_probs = torch.softmax(content_logits, dim=-1)
                    content_target_probs = content_probs[range(
                        content_logits.shape[0]), batch["content_labels"]]
                    adv_content_loss = torch.log(1 - content_target_probs).mean()
                    combined_loss = style_loss + 0.1 * adv_content_loss
                    total_combined_loss += combined_loss.item()

                    combined_loss = combined_loss / gradient_accumulation_steps
                    combined_loss.backward()

                    if (step + 1) % gradient_accumulation_steps == 0:
                        style_optimizer.step()
                        style_optimizer.zero_grad()
                        style_scheduler.step(epoch + step / len(train_dataloader))
                    progress_bar.update(1)

            avg_style_loss = total_style_loss / len(train_dataloader)
            avg_content_loss = total_content_loss / len(train_dataloader)
            avg_combined_loss = total_combined_loss / len(train_dataloader)
            writer.add_scalar('Loss/style_train', avg_style_loss, epoch)
            writer.add_scalar('Loss/content_train', avg_content_loss, epoch)
            writer.add_scalar('Loss/combined_train', avg_combined_loss, epoch)

            content_lrs = get_current_lr(content_optimizer)
            writer.add_scalar('lr/content_classifier', content_lrs[0], epoch)
            style_lrs = get_current_lr(style_optimizer)
            writer.add_scalar('lr/deberta', style_lrs[0], epoch)
            writer.add_scalar('lr/style_classifier', style_lrs[-1], epoch)

```

```
writer.close()

torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict().copy(),
    'content_optimizer_state_dict': content_optimizer.state_dict(),
    'style_optimizer_state_dict': style_optimizer.state_dict(),
}, f'runs/{run}/last_checkpoint.pth')

print('FINISHED')
```

A.2. KL Divergence Training Code

```
import os
from itertools import chain

os.environ["CUDA_VISIBLE_DEVICES"] = "1"

import sklearn
import evaluate
import torch
from datasets import load_dataset
from tensorboardX import SummaryWriter
from torch import nn
from torch.autograd import Function
from torch.optim import AdamW
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts
from torch.utils.data import DataLoader
from tqdm.auto import tqdm
from transformers import AutoModel, AutoTokenizer, DataCollatorWithPadding

data_files = {"train": "train.csv",
              "validation": "dev.csv", "test": "test.csv"}
raw_datasets = load_dataset("csv", data_files=data_files)
checkpoint = "microsoft/deberta-v3-base"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)


def tokenize_function(example):
    return tokenizer(example["Sentence"], truncation=True)


tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

tokenized_datasets = tokenized_datasets.remove_columns(
    ["Sentence Index", "Sentence", "Split"])
tokenized_datasets = tokenized_datasets.rename_column(
    "Style Label", "style_labels")
tokenized_datasets = tokenized_datasets.rename_column(
    "Content Label", "content_labels")
```

```python
tokenized_datasets.set_format("torch")

train_dataloader = DataLoader(
    tokenized_datasets["train"], shuffle=True, batch_size=16, collate_fn=data_collator
)
eval_dataloader = DataLoader(
    tokenized_datasets["validation"], batch_size=32, collate_fn=data_collator
)


class KLDivModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.deberta = AutoModel.from_pretrained(checkpoint)
        self.dropout = nn.Dropout(self.deberta.config.hidden_dropout_prob)
        self.style_classifier = nn.Linear(self.deberta.config.hidden_size, 2)
        self.content_classifier = nn.Linear(
            self.deberta.config.hidden_size, 208)

        # Freeze all parameters
        for param in self.deberta.parameters():
            param.requires_grad = False

        # Unfreeze the last layers
        for i in range(-1, -7, -1):
            for param in self.deberta.encoder.layer[i].parameters():
                param.requires_grad = True

    def forward(self, input_ids, attention_mask):
        outputs = self.deberta(input_ids, attention_mask=attention_mask)
        pooled_output = outputs.last_hidden_state[:, 0, :]
        pooled_output = self.dropout(pooled_output)
        style_logits = self.style_classifier(pooled_output)
        content_logits = self.content_classifier(pooled_output)
        return style_logits, content_logits


model = KLDivModel()

base_lr = 5e-5
decay_factor = 0.75

# Get the layers of the DeBERTa model
layers_to_train = model.deberta.encoder.layer[-1:-7:-1]

# Calculate learning rates for each layer
layerwise_lr = [base_lr * (decay_factor ** i) for i in range(len(layers_to_train))]

# Create parameter groups with the calculated learning rates
param_groups = []
for lr, layer in zip(layerwise_lr, layers_to_train):
    param_groups.append({'params': layer.parameters(), 'lr': lr})

# Add other model parameters (e.g., style_classifier) with the base learning rate
```

```python
param_groups.append({'params': model.style_classifier.parameters(), 'lr': base_lr*10})

content_optimizer = AdamW(model.content_classifier.parameters(), lr=base_lr*10, eps=1e-6)
style_optimizer = AdamW(param_groups)

T_0 = 6 # Number of epochs before the first restart
content_scheduler = CosineAnnealingWarmRestarts(content_optimizer, T_0=T_0)
style_scheduler = CosineAnnealingWarmRestarts(style_optimizer, T_0=T_0)


# Retrieve the current learning rates of the optimizer.
def get_current_lr(optimizer):
    lrs = [pg['lr'] for pg in optimizer.param_groups]
    return lrs


num_epochs = 120
num_training_steps = num_epochs * len(train_dataloader)

device = torch.device(
    "cuda") if torch.cuda.is_available() else torch.device("cpu")
print(device)
model.to(device)

progress_bar = tqdm(range(num_training_steps))

run = 'kl_div_training_1.18'

writer = SummaryWriter(log_dir=f'runs/{run}')

metrics = {
    "accuracy": evaluate.load("accuracy"),
    "f1": evaluate.load("f1"),
    "precision": evaluate.load("precision"),
    "recall": evaluate.load("recall"),
}

model.train()


def evaluate_model(model, dataloader, metrics):
    model.eval()
    total_style_loss = 0
    total_content_loss = 0
    style_preds = []
    style_labels = []
    content_preds = []
    content_labels = []

    with torch.no_grad():
        for batch in dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            style_logits, content_logits = model(
                batch["input_ids"], batch["attention_mask"])

            style_loss = nn.CrossEntropyLoss()(
                style_logits, batch["style_labels"])
            content_loss = nn.CrossEntropyLoss()(
                content_logits, batch["content_labels"])
            total_style_loss += style_loss.item()
            total_content_loss += content_loss.item()
            style_preds.extend(style_logits.argmax(dim=1).cpu().numpy())
            content_preds.extend(content_logits.argmax(dim=1).cpu().numpy())
            style_labels.extend(batch["style_labels"].cpu().numpy())
            content_labels.extend(batch["content_labels"].cpu().numpy())

    results = {}
    avg_style_loss = total_style_loss / len(dataloader)
    avg_content_loss = total_content_loss / len(dataloader)
    results['Loss/style'] = avg_style_loss
    results['Loss/content'] = avg_content_loss
    for name, metric in metrics.items():
        if (name == 'accuracy'):
            style_result = metric.compute(
                predictions=style_preds, references=style_labels)
            content_result = metric.compute(
                predictions=content_preds, references=content_labels)
        else:
            style_result = metric.compute(
                predictions=style_preds, references=style_labels, average='micro')
            content_result = metric.compute(
                predictions=content_preds, references=content_labels, average='micro')
        results[f'{name}/style'] = style_result[name]
        results[f'{name}/content'] = content_result[name]

    model.train()
    return results


gradient_accumulation_steps = 4

avg_style_loss = 0
avg_content_loss = 0
avg_combined_loss = 0

for epoch_block in range(0, num_epochs, 12): # Alternating training of objectives
    for objective in range(2):  # Two objectives: style and content
        for sub_epoch in range(6):  # Train each objective
            epoch = epoch_block + objective*6 + sub_epoch
            results = evaluate_model(model, eval_dataloader, metrics)
            print(f'Evaluation Results Epoch {epoch}:', results, flush=True)  # Print the
    results
            writer.add_scalar('Loss/style_val', results['Loss/style'], epoch)
            writer.add_scalar('Loss/content_val', results['Loss/content'], epoch)

            # Log metrics to TensorBoard
            for name, value in results.items():
                writer.add_scalar(f'{name}_val', value, epoch)

            total_style_loss = 0
            total_content_loss = 0
            total_combined_loss = 0

            if objective == 0:
                # Content classifier update
                for step, batch in enumerate(train_dataloader):
                    batch = {k: v.to(device) for k, v in batch.items()}
                    input_ids = batch["input_ids"]
                    attention_mask = batch["attention_mask"]
                    style_logits, content_logits = model(
                        input_ids=input_ids, attention_mask=attention_mask)
                    content_loss = nn.CrossEntropyLoss()(
                        content_logits, batch["content_labels"])
                    style_loss = nn.CrossEntropyLoss()(
                        style_logits, batch["style_labels"])
                    total_content_loss += content_loss.item()
                    total_style_loss += style_loss.item()

                    # KL divergence loss for content
                    content_probs = log_softmax(content_logits, dim=-1)
                    # Create a uniform distribution tensor of the same size as content_logits
                    num_classes = content_logits.size(-1)
                    uniform_distribution = torch.full_like(content_probs, 1.0 / num_classes).to(device)
                    # Compute the KL divergence
                    kl_div_content_loss = kl_div(content_probs, uniform_distribution, reduction='batchmean')
                    combined_loss = style_loss + kl_div_content_loss
                    total_combined_loss += combined_loss.item()

                    content_loss = content_loss / gradient_accumulation_steps
                    content_loss.backward()

                    if (step + 1) % gradient_accumulation_steps == 0:
                        content_optimizer.step()
                        content_optimizer.zero_grad()
                        content_scheduler.step(epoch + step / len(train_dataloader))
                    progress_bar.update(1)

            else:
                # Style classifier and DeBERTa update
                for step, batch in enumerate(train_dataloader):
                    batch = {k: v.to(device) for k, v in batch.items()}
                    input_ids = batch["input_ids"]
                    attention_mask = batch["attention_mask"]
                    style_logits, content_logits = model(
                        input_ids=input_ids, attention_mask=attention_mask)
                    style_loss = nn.CrossEntropyLoss()(
                        style_logits, batch["style_labels"])
                    content_loss = nn.CrossEntropyLoss()(
                        content_logits, batch["content_labels"])
                    total_style_loss += style_loss.item()
                    total_content_loss += content_loss.item()

                    # KL divergence loss for content
                    content_probs = log_softmax(content_logits, dim=-1)
                    # Create a uniform distribution tensor of the same size as content_logits
                    num_classes = content_logits.size(-1)
                    uniform_distribution = torch.full_like(content_probs, 1.0 / num_classes).to(device)
                    # Compute the KL divergence
                    kl_div_content_loss = kl_div(content_probs, uniform_distribution, reduction=’batchmean’)
                    combined_loss = style_loss + kl_div_content_loss
                    total_combined_loss += combined_loss.item()

                    combined_loss = combined_loss / gradient_accumulation_steps
                    combined_loss.backward()

                    if (step + 1) % gradient_accumulation_steps == 0:
                        style_optimizer.step()
                        style_optimizer.zero_grad()
                        style_scheduler.step(epoch + step / len(train_dataloader))
                    progress_bar.update(1)

            avg_style_loss = total_style_loss / len(train_dataloader)
            avg_content_loss = total_content_loss / len(train_dataloader)
            avg_combined_loss = total_combined_loss / len(train_dataloader)
            writer.add_scalar(’Loss/style_train’, avg_style_loss, epoch)
            writer.add_scalar(’Loss/content_train’, avg_content_loss, epoch)
            writer.add_scalar(’Loss/combined_train’, avg_combined_loss, epoch)

            content_lrs = get_current_lr(content_optimizer)
            writer.add_scalar(’lr/content_classifier’, content_lrs[0], epoch)
            style_lrs = get_current_lr(style_optimizer)
            writer.add_scalar(’lr/deberta’, style_lrs[0], epoch)
            writer.add_scalar(’lr/style_classifier’, style_lrs[-1], epoch)

writer.close()

torch.save({
    ’epoch’: epoch,
    ’model_state_dict’: model.state_dict().copy(),
    ’content_optimizer_state_dict’: content_optimizer.state_dict(),
    ’style_optimizer_state_dict’: style_optimizer.state_dict(),
}, f’runs/{run}/last_checkpoint.pth’)

print(’FINISHED’)
```

A.3. Prompting Techniques

System Prompt LLaMA 2

You are an automated rephrasing tool. You produce accurate outputs and follow the user's instructions precisely. You use words for math ("plus" instead of "+", "divided by" instead of "/" etc.).

System Prompt Vicuna

A chat between a user and an automated rephrasing tool. The tool produces accurate outputs and follows the user's instructions precisely. The tool uses words for math ("plus" instead of "+", "divided by" instead of "/" etc.).

Prompting Techniques

Zero-shot:
Rephrase the following text, delimited by ```, such that the rephrased text has improved structure and fluency. The rephrased text is to be semantically equivalent to the original text.

Original text:
```
{text}
```

Zero-shot with Original Context:
Rephrase the following text, delimited by ``` such that the rephrased text has improved structure and fluency. The rephrased text is to be semantically equivalent to the original text.

Context (the three original sentences before the original text):
```
{context}
```

Original text:
```
{text}
```

Prompting Techniques

Zero-shot with Rephrased Context:

~~~
Rephrase the following text, delimited by ``` such that the rephrased text has improved structure and fluency. The rephrased text is to be semantically equivalent to the original text.

Context (the three rephrased sentences before the original text):
```
{context}
```

Original text:
```
{text}
```
~~~

Few-shot:

~~~
Rephrase the following text, delimited by ``` such that the rephrased text has improved structure and fluency. The rephrased text is to be semantically equivalent to the original text.

Original text:
```
{text}
```
~~~

Interactions for Proper Output Format

The following describes the interaction format for ensuring proper output format for parsing. It only outlines the general interactions, the concrete realization of this then depends on the official prompt format recommended for the respective model.

Zero-shot:

USER: {instruction}

{context}

Original text:
```
Example text.
```

ASSISTANT: Rehrased text:
```
Example text.
```

USER: Original text:
```
{text}
```

Few-shot:

Same approach as zero-shot (without context), but instead of “Example text.” use the sentences from below for the first three interactions.

Sentences for Few-shot

Original sentences:

1. I mean, so I just took the logs from the task and I compared our system with them, this was done after the fact so we didn't participate in IWCD 2020 this is just our system compared with the three with the three submissions.
2. So what you're doing then is, if you have my dog is cute, he likes to play, he likes playing, you're putting in, you can put in the whole input, and you have some extra tokens, so you put an extra separator between the sentence.
3. Techniques, there's like a, yeah, I could give like one complete lecture only about like how to smooth things.

Rephrased sentences:

1. I took the logs from the task and compared our system with them. This was done after the fact since we didn't participate in IWCD 2020. Now, this is our system compared with the three submissions.
2. What you're doing then is, if you have sentences like "My dog is cute. He likes playing.", you can put in the whole input, and since you have some extra tokens, you put a separator between the sentences.
3. Techniques, I could give one complete lecture only on how to smooth things.

Prompt for Single Retry on Parsing Error

This does not seem right. Please only output the rephrased text, wrapped by ```.